7.5

Anwendungen für IBM WebSphere MQ entwickeln

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 1187 gelesen werden.

Diese Ausgabe bezieht sich auf Version 7 Release 5 von IBM® WebSphere MQ und auf alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

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

Inhaltsverzeichnis

Anwendungen entwickeln

IBM WebSphere MQ bietet mehrere Möglichkeiten, mit denen Sie Anwendungen entwickeln können, um Nachrichten zur Unterstützung Ihrer Geschäftsprozesse zu senden und zu empfangen. Darüber hinaus können Sie Anwendungen für das Management Ihrer Warteschlangenmanager und zugehörigen Ressourcen entwickeln.

Bevor Sie mit dem Entwickeln von Anwendungen für IBM WebSphere MQ beginnen, müssen Sie mit den Konzepten im Thema IBM WebSphere MQ - Technische ÜbersichtIBM WebSphere MQ - Technische Übersicht vertraut sein.

Sie können Anwendungen für IBM WebSphere MQ in vielen verschiedenen Programmiersprachen entwickeln. Informationen zu den unterstützten Programmiersprachen und deren Features finden Sie im Thema „Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82.

In den folgenden Abschnitten werden die Anwendungstypen beschrieben, die Sie für IBM WebSphere MQ auf verschiedenen Plattformen schreiben können.

Typen von Anwendungen, die Sie für IBM WebSphere MQ

Hier werden die Anwendungstypen aufgeführt, die unter IBM WebSphere MQerstellt werden können.

Bei IBM WebSphere MQ-Produkten handelt es sich um Warteschlangenmanager und Anwendungsenabler. Sie unterstützen die IBM Message Queue Interface (MQI), über die Programme Nachrichten in eine Warteschlange einreihen und von dieser abrufen können.

Mit IBM WebSphere MQ für andere als z/OS-Plattformen können Sie Anwendungen schreiben, die:

- 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 WebSphere MQ-Plattformen ausgeführt werden.
- Message-Queuing aus CICS for TXSeries for AIX-, TXSeries for HP-UX-, TXSeries for Solaris- und TXSeries for Windows-Systemanwendungen verwenden.
- Message-Queuing aus Encina for AIX-, HP-UX-, Solaris- und Windows-Systemen verwenden.
- Verwenden Sie die Steuerung von Nachrichtenwarteschlangen aus Tuxedo für AIX-, AT & T-, HP-UX-, Solaris-und Windows -Systeme.
- Mit IBM WebSphere MQ als Transaktionsmanager Aktualisierungen koordinieren, die externe Ressourcenmanager inIBM WebSphere 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.
- Von einer vollständigen IBM WebSphere MQ-Umgebung oder einer IBM WebSphere MQ-MQI-Client-Umgebung aus auf den folgenden Plattformen ausgeführt werden:
  - UNIX and Linux®-Systeme
  - Windows

Zugehörige Konzepte

Sicherheit

Konzepte für die Anwendungsentwicklung

Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

Bevor Sie beginnen, Ihre IBM WebSphere MQ -Anwendungen zu entwerfen und zu schreiben, machen Sie sich mit den grundlegenden IBM WebSphere MQ -Konzepten vertraut. Lesen Sie hierzu die Themen unter Technische Übersicht. Informationen zu den Anwendungstypen, die Sie für IBM WebSphere MQschreiben können, finden Sie unter „Anwendungen entwickeln” auf Seite 7.

Verwenden Sie die folgenden Links, um Informationen zu den IBM WebSphere MQ -Konzepten für die Anwendungsentwicklung zu erhalten:

- „IBM WebSphere MQ-Nachrichten” auf Seite 10
- Punkt-zu-Punkt-Messaging
- Einführung in das Publish/Subscribe-Messaging von WebSphere MQ
- „Message Queue Interface (MQI) in Clientanwendungen verwenden” auf Seite 375
- „Web-Services in WebSphere MQ verwenden” auf Seite 1004
- „Kanalexitprogramme für Messaging-Kanäle” auf Seite 422
- „Transaktionsunterstützungsszenarios” auf Seite 43

Bevor Sie Anwendungen ausführen können, die mit der MQI arbeiten, müssen Sie bestimmte IBM WebSphere MQ-Objekte erstellen. Weitere Informationen finden Sie im Thema „Anwendungsprogramme, die die MQI verwenden” auf Seite 9.

Zugehörige Konzepte

„IBM WebSphere MQ-Anwendungen entwerfen” auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„WebSphere MQ-Beispielprogramme” auf Seite 100
In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

„Anwendung zur Warteschlangensteuerung schreiben” auf Seite 206
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.

„Clientanwendungen schreiben” auf Seite 374
Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„WebSphere MQ-Klassen für JMS verwenden” auf Seite 759
WebSphere MQ Classes for Java Message Service (WebSphere MQ Classes for JMS) ist der JMS-Anbieter, der mit WebSphere MQbereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die WebSphere MQ-Klassen für JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„Component Object Model-Schnittstelle verwenden (WebSphere MQ Automation Classes for ActiveX)” auf Seite 1094
WebSphere MQ Automation Classes for ActiveX (MQAX) sind ActiveX-Komponenten, die Klassen bereitstellen, die Sie in Ihrer Anwendung für den Zugriff auf WebSphere MQ verwenden können.

„WebSphere MQ Classes for Java verwenden” auf Seite 693

Mit WebSphere MQ Classes for Java können Sie WebSphere MQ in einer Java-Umgebung verwenden. Eine Java-Anwendung kann entweder WebSphere MQ Classes for Java oder WebSphere MQ Classes for JMS verwenden, um auf WebSphere MQ -Ressourcen zuzugreifen.

„.NET verwenden" auf Seite 594
WebSphere MQ-Klassen für .NET ermöglichen es einem Programm, das im .NET-Programmierungsframework geschrieben wurde, eine Verbindung zu WebSphere MQ als WebSphere MQ-MQI-Client herzustellen oder sich direkt mit einem WebSphere MQ-Server zu verbinden.

„C++ verwenden" auf Seite 668
WebSphere MQ bietet C++-Klassen, die WebSphere MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die den Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

„IBM WebSphere MQ-Anwendung erstellen" auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

Anwendungsprogramme, die die MQI verwenden

Zur erfolgreichen Ausführung benötigen IBM WebSphere MQ-Anwendungsprogramme bestimmte Objekte.

Abbildung 1 auf Seite 9 zeigt eine Anwendung, die Nachrichten aus einer Warteschlange entfernt, sie verarbeitet und anschließend einige Ergebnisse an eine andere Warteschlange im selben Warteschlangenmanager sendet.

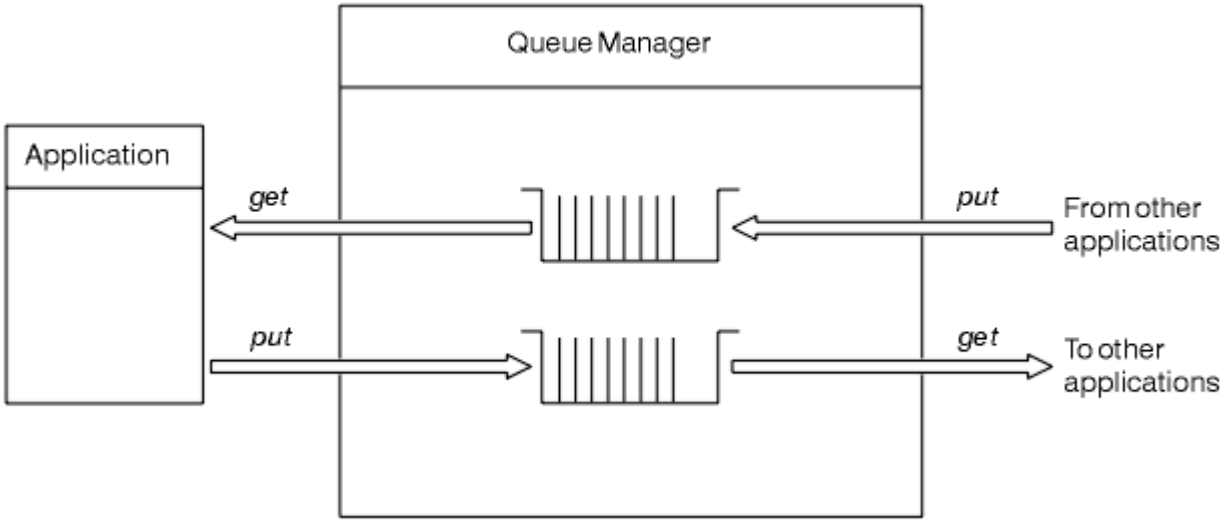


Abbildung 1. Warteschlangen, Nachrichten und Anwendungen

Anwendungen können Nachrichten zwar in lokale oder ferne Warteschlangen einreihen (mittels MQPUT), direkt abrufen können sie sie allerdings nur von lokalen Warteschlangen (mittels MQGET).

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 WebSphere MQ erforderlich. (siehe „IBM WebSphere MQ-Anwendung erstellen" auf Seite 456).
- 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 9 nicht dargestellt.

IBM WebSphere MQ-Nachrichten

In diesem Abschnitt werden Konzept, Teile und Deskriptor von IBM WebSphere MQ-Nachrichten erläutert.

IBM WebSphere MQ-Nachrichten bestehen aus zwei Teilen:

- Nachrichteneigenschaften
- Anwendungsdaten

Abbildung 2 auf Seite 10 stellt eine Nachricht dar und zeigt deren logische Aufteilung in Nachrichteneigenschaften und Anwendungsdaten.

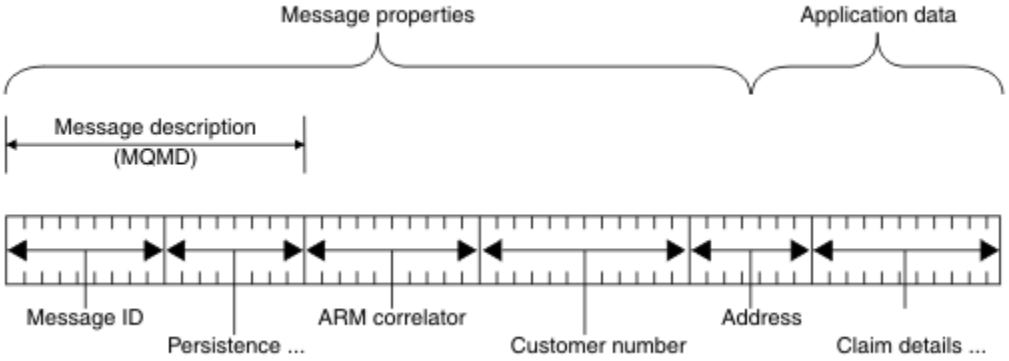


Abbildung 2. Darstellung einer Nachricht

Die in einer WebSphere MQ-Nachricht enthaltenen Anwendungsdaten werden vom Warteschlangenmanager nicht geändert, es sei denn, es erfolgt eine Datenkonvertierung. Auch der Dateninhalt unterliegt in WebSphere MQ keinerlei Beschränkung. Die Datenlänge der einzelnen Nachrichten darf die Werte des Attributs *MaxMsgLength* der Warteschlange und des Warteschlangenmanagers nicht überschreiten.

Unter WebSphere MQ for AIX, WebSphere MQ for HP-UX, WebSphere MQ for Linux, WebSphere MQ for Solaris, und WebSphere MQ für Windowsist der Standardwert für *MaxMsgLength* 100 MB (104 857 600 Byte).

Unter gewissen Umständen sollten Ihre Nachrichten etwas kürzer als *MaxMsgLength* sein. Weitere Informationen finden Sie im Abschnitt „Die Daten der Nachricht” auf Seite 244.

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

Über die MQMD-Struktur, die den *Nachrichtendeskriptor* definiert, ist ein Zugriff auf die Steuerinformationen von Nachrichten möglich.

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

Im Abschnitt „Nachrichtenkontext” auf Seite 40 wird erläutert, wie die MQMD-Felder mit Informationen zum Ursprung der Nachricht verwendet werden können.

Es gibt verschiedene Versionen des Nachrichtendeskriptors. Weitere Informationen zur Gruppierung und Segmentierung von Nachrichten (siehe „Nachrichtengruppen” auf Seite 37) erhalten Sie in Version 2 des Nachrichtendeskriptors (oder in MQMDE). Dieser ist mit dem Nachrichtendeskriptor der Version 1 identisch, verfügt jedoch über zusätzliche Felder. Eine Beschreibung dieser Felder finden Sie im Thema MQMDE - Nachrichtendeskriptorerweiterung.

Nachrichtentypen

In IBM WebSphere 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.IBM WebSphere MQ definiert einen Bereich von Rückkopplungscodes für Anwendungen, die verwendet werden sollen.

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.)

Berichte und segmentierte Nachrichten

Wird unter WebSphere MQ for z/OS nicht unterstützt.

Wenn eine Nachricht segmentiert ist (eine Beschreibung segmentierter Nachrichten finden Sie in Abschnitt „Nachrichtensegmentierung" auf Seite 279) und Sie die Generierung von Berichten anfordern, erhalten Sie unter Umständen mehr Berichte als bei einer nicht segmentierten Nachricht.

Für von WebSphere MQ generierte Berichte

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 WebSphere MQ-Header am Anfang der Daten der ursprünglichen Nachricht 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 WebSphere 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 geben diese WebSphere 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 WebSphere 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 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 ursprünglichen Nachrichtendaten ohne die Längen aller WebSphere MQ -Header, die Sie finden.

Berichte abrufen

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

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

Sie können dieses Verfahren auch anwenden, wenn mehrere verschiedene Berichtstypen in der Warteschlange vorhanden sind (z. B. COA und COD), da ein MQGET mit MQGMO_COMPLETE_MSG Berichtsnachrichten nur dann neu erstellt, wenn sie denselben *Feedback* -Code haben. Sie können dieses Verfahren jedoch normalerweise nicht für Ausnahmeberichte verwenden, da diese im Allgemeinen 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. Sie können das Feld *GroupId* in der Berichtsnachricht verwenden, um Berichte mit dem*GroupId* der ursprünglichen Nachricht zu korrelieren, und das Feld *Feedback* verwenden, um den Typ jeder Berichtsnachricht anzugeben. 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 Bericht von einem Warteschlangenmanager generiert wird, der die Segmentierung unterstützt, aber auf einem Warteschlangenmanager empfangen wird, der *keine* Segmentierung unterstützt, ist die MQMDE-Struktur (die die *Offset* und *OriginalLength* angibt, die durch den Bericht dargestellt werden) immer in den Berichtsdaten enthalten, zusätzlich zu null, 100 Byte oder allen ursprünglichen Daten in der Nachricht.

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)
- WebSphere MQ-definierte Nachrichten, z. B. Programmable Command Formats

  WebSphere 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 Formaten müssen die Anwendungsdaten von einem Datenkonvertierungsexitprogramm konvertiert werden (weitere Informationen finden Sie im Thema „Datenkonvertierungsexits schreiben" auf Seite 442). 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

WebSphere 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 WebSphere MQ den SBCS-Teil der Seite als Warteschlangenmanager-CCSID.

Wenn das zugrunde liegende Betriebssystem Seiten mit DBCS-Code unterstützt, kann WebSphere 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 442.

Nachrichtenprioritäten

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 Warteschlangendefinition enthält (siehe „Namensauflösung” auf Seite 230), wird zum Zeitpunkt der Put-Operation die Standardpriorität von dem Wert des Attributs *DefPriority* der im Befehl 'Open' angegebenen Warteschlange verwendet.

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 WebSphere MQ hat das Attribut den Wert 9. Sie können Nachrichten mit Prioritäten 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. Zudem vereinfachen Nachrichteneigenschaften die Kommunikation zwischen WebSphere 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 WebSphere MQ imitiert die Verwendung von Eigenschaften in JMS. Sie können also Eigenschaften in einer JMS-Anwendung festlegen und sie in einer prozeduralen WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

Nachrichteneigenschaften, die an einen Warteschlangenmanager einer älteren Version des Produkts als Version 7.0 geleitet werden, zählen zur Länge der Nachricht hinzu. Hiervon ausgenommen sind Nachrichteneigenschaften im Nachrichtendeskriptor. Erhöhen Sie also entsprechend den Wert des Attributs *MaxMsgLength* von Kanälen, die in ein System vor Version 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 in Java Message Service (JMS) keine maximale Länge für den Eigenschaftsnamen festgelegt wird, kann eine JMS-Anwendung einen gültigen JMS-Eigenschaftsnamen festlegen, der beim Speichern in einer MQRFH2-Struktur jedoch nicht als gültiger WebSphere MQ-Eigenschaftsname gilt.

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.

Alle Eigenschaftsnamen müssen den Regeln entsprechen, die in der Java Language Specification for Java Identifiers definiert sind, mit der Ausnahme, dass das Unicode-Zeichen U+002E (.) als Teil des Namens zulässig ist-aber nicht der Anfang. Die Regeln für Java-Kennungen 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.

WebSphere 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 Benennungskonvention 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" - für die Verwendung durch WebSphere MQ-Klassen für Java Message Service reserviert.
   - "usr.JMS" - nicht gültig.

   Ausgenommen hiervon sind lediglich die folgenden Eigenschaften, die Synonyme für JMS-Eigenschaften bereitstellen:

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

   Diese Synonyme ermöglichen einer MQI-Anwendung den Zugriff auf JMS-Eigenschaften auf ähnliche Weise wie bei einer Clientanwendung mit WebSphere MQ-Klassen für Java Message Service. Von diesen Eigenschaften können nur JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID und JMSXGroupSeq mithilfe von MQI festgelegt werden.

   Beachten Sie, dass die Eigenschaften des Typs JMS_IBM_*, die in den WebSphere MQ-Klassen für Java Message Service zur Verfügung stehen, über die 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 sind die Namen der SQL-Schlüsselwörter, die in Auswahlzeichenfolgen verwendet werden.
3. Ein Eigenschaftsname, der mit "mq " beginnt. In jeder Mischung aus Klein-oder Großbuchstaben und nicht beginnend mit "mq_usr" kann nur ein "." Zeichen (U+002E). Mehrere "." Zeichen sind in Eigenschaften mit diesen Präfixen nicht zulässig.
4. Zwei "." Zeichen müssen andere Zeichen dazwischen enthalten; ein leerer Punkt in der Hierarchie ist nicht zulässig. Ebenso darf ein Eigenschaftsname nicht mit einem "." enden. .
5. Wenn eine Anwendung die Eigenschaft "a.b" und dann die Eigenschaft "a.b.c" festlegt, ist unklar, ob "b" in der Hierarchie einen Wert oder eine andere logische Gruppierung enthält. Eine solche Hierarchie ist "gemischter Inhalt" und dies wird nicht unterstützt. Die Festlegung einer Eigenschaft, die zu gemischtem Inhalt führt, ist nicht zulässig.

Diese -Einschränkungen werden vom Validierungsmechanismus wie folgt umgesetzt:

- Eigenschaftsnamen werden validiert, wenn eine Eigenschaft mit dem Aufruf MQSETMP - Nachrichteneigenschaft einrichten festgelegt wird, wenn die Validierung angefordert wurde, als die Nachrichtenkennung erstellt wurde. Wenn ein Versuch unternommen wird, eine Eigenschaft zu validieren, und dieser Versuch aufgrund eines Fehlers in der Spezifikation des Eigenschaftsnamens fehlschlägt, lautet der Beendigungscode MQCC_FAILED mit folgender Ursache:
  - 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 derselben Groß-/Kleinschreibung 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. Weitere Informationen finden Sie im Abschnitt "MQMDE angegeben in MQPUT- und MQPUT1-Aufrufen" im Thema Übersicht zu 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 Bytefolgeeigenschaften.

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.

Die Angabe einer ID des codierten Zeichensatzes, für die keine Unicode-Konvertierung von WebSphere MQ unterstützt wird, führt zu dem 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 25 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQSUB.

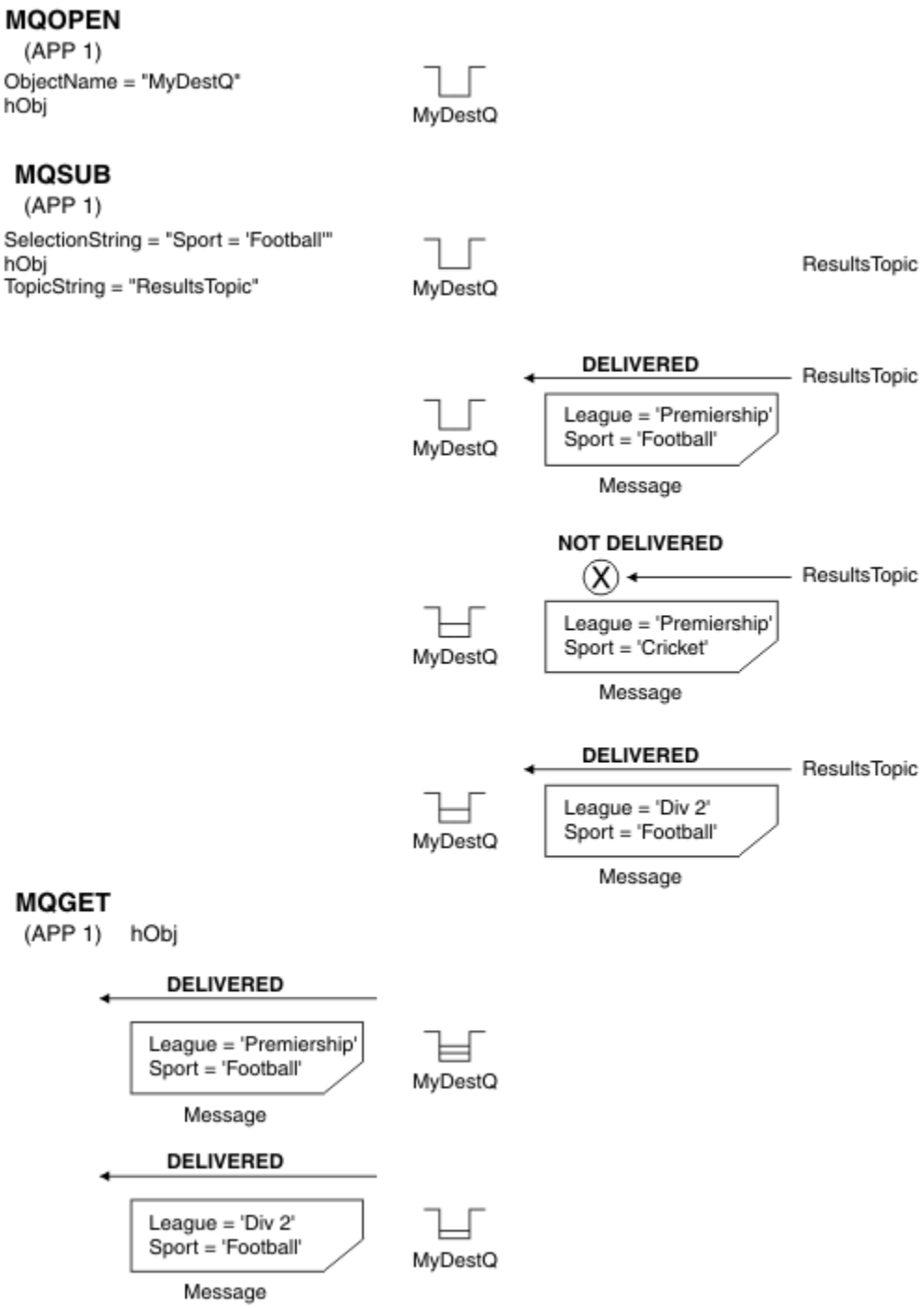


Abbildung 3. Auswahl mithilfe des Aufrufs MQSUB

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 26 zeigt den Auswahlprozess unter Verwendung des Aufrufs MQOPEN.

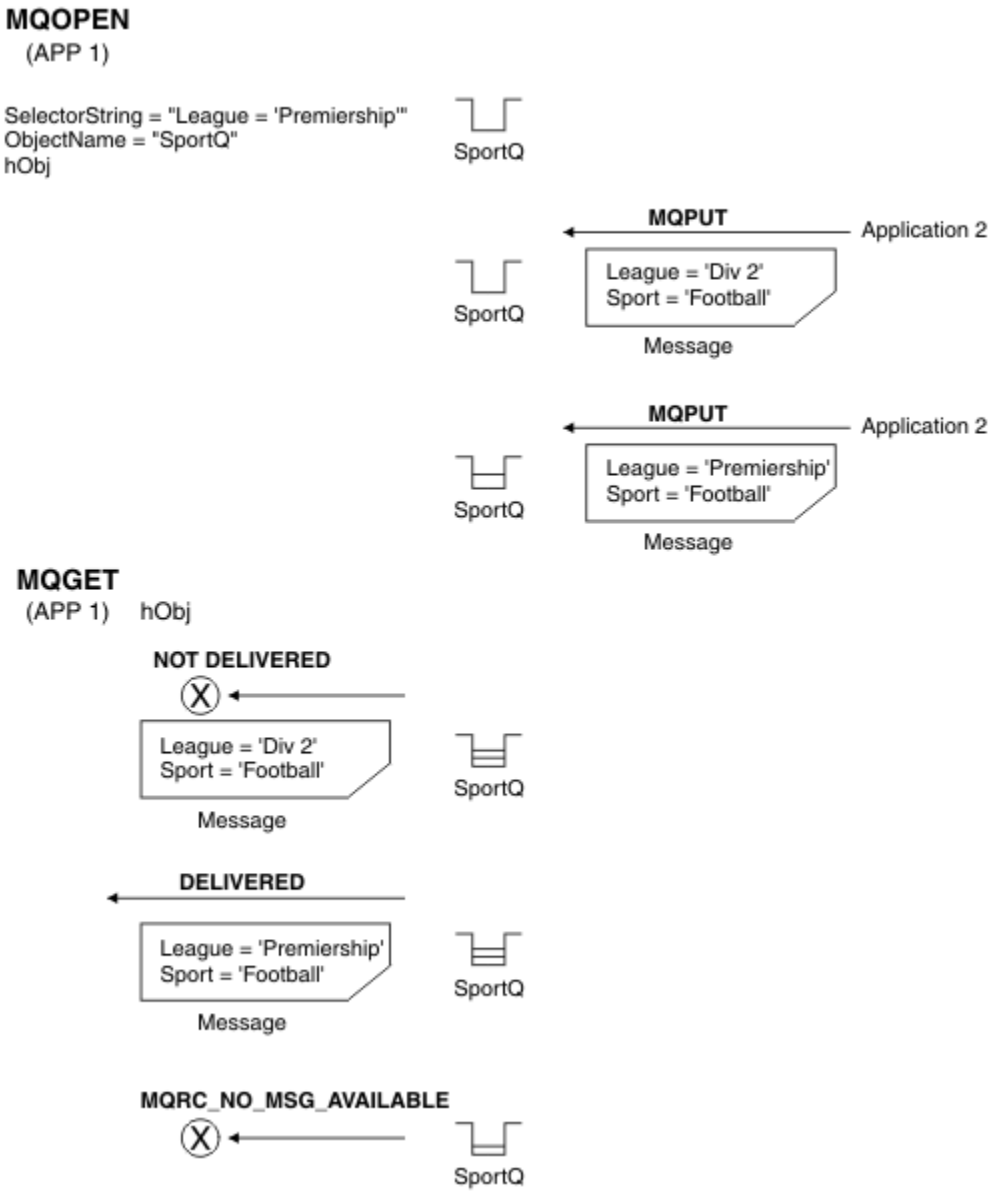


Abbildung 4. Auswahl mithilfe des Aufrufs MQOPEN

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.

Auswahlverhalten
Übersicht über das Auswahlverhalten von IBM WebSphere MQ .

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 ist möglich, dass eine Auswahlzeichenfolge in TRUE oder FALSE aufgelöst wird, bevor ein Abgleich mit Nachrichteneigenschaften erfolgt. Dies ist beispielsweise der Fall, 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.

Eine Nachricht kann beispielsweise eine Auswahlzeichenfolge wie `'Color IS NULL'`erfüllen, wobei `'Color'` nicht als Nachrichteneigenschaft in der Nachricht vorhanden ist.

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 WebSphere 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 in tiefen Warteschlangen zu optimieren, wenn die Auswahl auf JMSCorrelationID oder JMSMessageIDbasiert, verwenden Sie eine Auswahlzeichenfolge im Format `JMSCorrelationID = ...` oder `JMSMessageID = ...` und verweisen Sie nur auf eine Eigenschaft.

Diese Methode ermöglicht eine wesentliche Leistungssteigerung bei der Auswahl mit 'JMSCorrelationID' und eine marginale Leistungssteigerung für 'JMSMessageID'.

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 WebSphere MQ das System, indem die Verarbeitung zu komplexer Selektoren fehlschlägt, da andernfalls eine Knappheit von Systemressourcen entstünde. Schutz kann nach ungefähr 100 Tests auf einigen Plattformen erfolgen, weshalb Selektoren, die sich dieser Zahl von Komponenten annähern, fehlschlagen können. Es wird empfohlen, die Verwendung von Auswahlzeichenfolgen mit mehreren Komponenten auf den entsprechenden Plattformen sorgfältig zu testen, 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

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

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 WebSphere MQ vorgenommen werden; stattdessen ist ein Provider für erweiterte Nachrichtenauswahl zur Verarbeitung der Nachrichten erforderlich, beispielsweise IBM Integration Bus.

Syntax der Nachrichtenselektoren
Ein WebSphere 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.

WebSphere MQ prüft die Richtigkeit der Syntax eines Nachrichtenselektors, wenn dieser dargestellt wird. Ist die Syntax der Auswahlzeichenfolge falsch oder ein Eigenschaftsname nicht gültig und kein erweiterter Nachrichtenauswahlbereitsteller verfügbar, 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.

Ein Selektor kann Folgendes enthalten:

- Literale:
  - Zeichenfolgeliterale stehen zwischen einfachen Anführungszeichen. Zwei aufeinanderfolgende einfache Anführungszeichen (Hochkommas) stellen ein einfaches Anführungszeichen dar, Beispiele: 'lite-

ral' und 'literal''s'. Wie Java-Zeichenfolgeliterale verwenden diese die Unicode-Zeichencodierung. Ein Zeichenfolgeliteral darf nicht in Anführungszeichen stehen. Zwischen einfachen Anführungszeichen darf jede Abfolge von Bytes verwendet werden.

– Eine Bytefolge besteht aus mindestens einem Paar Hexadezimalzeichen in Anführungszeichen und hat das Präfix `0x`. Beispiele sind `"0x2F1C"` oder `"0XD43A"`. Die Länge einer Bytefolge muss mindestens ein Byte betragen. Wenn eine Selektorbytefolge an eine Nachrichteneigenschaft vom Typ `MQTYPE_BYTE_STRING` angepasst wird, wird keine besondere Aktion an einer führenden oder abschließenden Null vorgenommen. Die Bytes werden als weiteres Zeichen behandelt. Endianness wird ebenfalls nicht berücksichtigt. Die Länge von Selektor- und Eigenschaftsbytefolgen muss identisch sein, ebenso wie die Bytefolge.

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

  `- "myBytes = "0x0AFC23"" = TRUE`

  Folgende Zeichenfolgenauswahlen stimmen nicht überein:

  - `"myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR` (die Anzahl der Bytes ist kein Vielfaches von zwei)
  - `"myBytes = "0x0AFC2300"" = FALSE` (die abschließende Null ist im Vergleich signifikant)
  - `"myBytes = "0x000AFC23"" = FALSE` (die führende Null ist im Vergleich signifikant)
  - `"myBytes = "0x23FC0A"" = FALSE` (Endianness wird nicht berücksichtigt)

– Hexadezimalzahlen beginnen mit einer Null, gefolgt von x in Groß- oder Kleinschreibung. Der Rest des Literals enthält ein oder mehrere gültige Hexadezimalzeichen. Beispiele: `0xA`, `0xAF`, `0X2020`.

– Eine führende Null, gefolgt von mindestens einer Ziffer zwischen 0-7, wird immer als Beginn einer Oktalzahl interpretiert. Sie können keine Dezimalzahl mit dem Präfix Null darstellen. `09` gibt beispielsweise einen Syntaxfehler zurück, da 9 keine gültige oktale Ziffer ist. Oktalzahlen sind beispielsweise `0177`, `0713`.

– An exact numeric literal is a numeric value without a decimal point, such as 57, `-957`, and +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. WebSphere MQ unterstützt exakte Ziffern im Bereich 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`. WebSphere MQ unterstützt Zahlen im Bereich `-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 in den Themen „Eigenschaftsnamen” auf Seite 20 und „Einschränkungen bei Eigenschaftsnamen” auf Seite 21.

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. WebSphere 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 860. Beispiel: Die Auswahlzeichenfolge `"JMSPriority >="` wählt die Eigenschaft `Pri` im Ordner `jms` der aktuellen Nachricht aus.

- Überlauf/Unterlauf:

  Folgendes ist für dezimale und ungefähre numerische Ziffern 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. Eine Ausnahme besteht darin, dass exakte numerische Werte und ungefähre numerische Werte verglichen werden können (die erforderliche Typenkonvertierung wird durch die Regeln der numerischen Java-Umstufung 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 numerische Java-Umstufung verwenden.
- 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. Sie entspricht dem Ausdruck `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` gibt 'False' für `'UK'` und 'True' für `'Peru'` zurück. Sie entspricht dem 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, aber als falsch ausgewertet. 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
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

Tabelle 2. Falls logisch, lautet das Ergebnis der booleschen Operatoren A OR B (Forts.)		
Operator A	**Operator B**	**Ergebnis (A OR B)**
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

Auswahlverhalten
Übersicht über das Auswahlverhalten von IBM WebSphere MQ .

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 WebSphere MQ vorgenommen werden; stattdessen ist ein Provider für erweiterte Nachrichtenauswahl zur Verarbeitung der Nachrichten erforderlich, beispielsweise IBM Integration Bus.

„Nachrichteneigenschaften” auf Seite 18
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. Zudem vereinfachen Nachrichteneigenschaften die Kommunikation zwischen WebSphere MQ- und JMS-Anwendungen.

Zugehörige Verweise

MsgHandle

MQBUFMH - Konvertieren von Puffern in Nachrichtenkennungen

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

- Die Äquivalenz wird mit einem einzelnen Gleichheitszeichen getestet. Beispiel: a = b ist richtig, während a == b falsch ist.
- Ein Operator, mit dem viele Programmiersprachen 'ungleich' darstellen, ist !=. Diese Darstellung ist kein gültiges Synonym für <>; beispielsweise ist a <> b gültig, während a != b ungültig ist.

- 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 logische Konjunktion/Disjunktion. Beispiel: `a && b` muss 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. Daher sind die Auswahlzeichenfolgen `a IS 3` und `b IS 'red'` nicht gültig. Das Schlüsselwort `IS` ist nur zur Unterstützung von `IS NULL` -und `IS NOT NULL` -Fällen vorhanden.

Zugehörige Konzepte

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

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.

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.

Zugehörige Konzepte

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.

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 WebSphere MQ vorgenommen werden; stattdessen ist ein Provider für erweiterte Nachrichtenauswahl 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, fordert WebSphere MQ, dass der Provider

für erweiterte Nachrichtenauswahl die Syntax der Veröffentlichung analysiert und WebSphere 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 erweiterte Nachrichtenauswahl fest, dass die Veröffentlichung nicht mit den Auswahlkriterien übereinstimmt, wird sie nicht an den Subskribenten übermittelt. 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 Konzepte

Auswahlverhalten
Übersicht über das Auswahlverhalten von IBM WebSphere MQ .

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

Asynchrone Verarbeitung von IBM WebSphere 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 36 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.

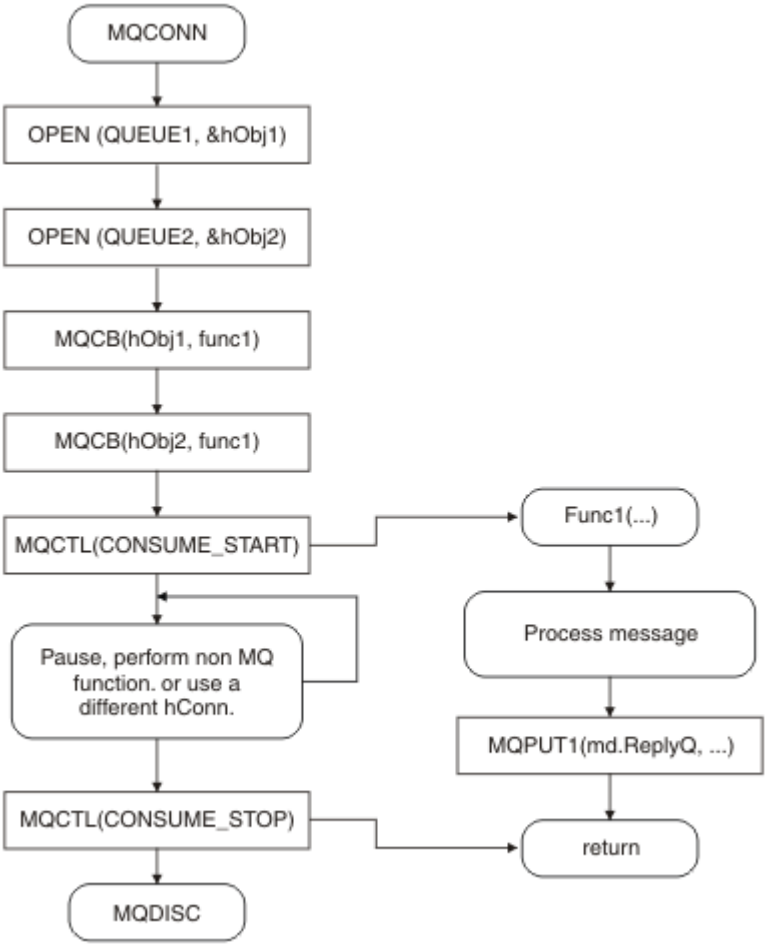


Abbildung 5. Nachrichtengesteuerte Standardanwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

Abbildung 6 auf Seite 37Dieser 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.

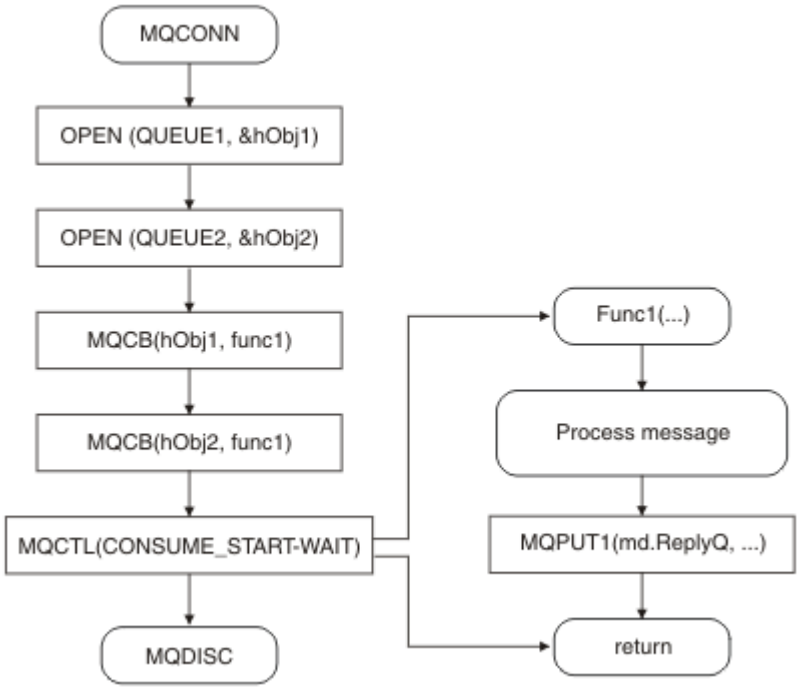


Abbildung 6. Nachrichtengesteuerte Einzelthread-Anwendung, die Nachrichten aus zwei Warteschlangen verarbeitet

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 261). Auf anderen Plattformen als z/OSermöglicht „Nachrichtensegmentierung" auf Seite 279 ein zugehöriges Konzept, dass große Nachrichten in kleinere Segmente aufgeteilt werden können. Sie können gruppierte oder segmentierte Nachrichten nicht verwenden, wenn Sie sie in ein Thema einreihen.

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 37 zeigt eine Gruppe logischer Nachrichten:

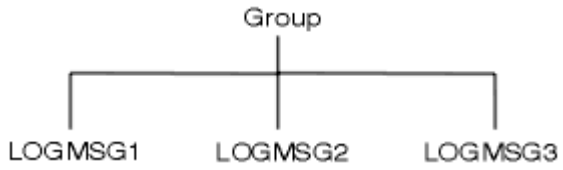


Abbildung 7. Gruppe logischer Nachrichten

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 logisch gesehen eine separate Nachricht und nur die Felder *GroupId* und *MsgSeqNumber* im MQMD müssen mit anderen Nachrichten in der Gruppe in Beziehung stehen. Andere Felder im MQMD sind unabhängig; einige können für alle Nachrichten in der Gruppe identisch sein, wogegen andere unterschiedlich sein können. Nachrichten können z. B. in einer Gruppe verschiedene Formatnamen, CCSIDs und Codierungen tragen.

Segment

Segmente werden für die Bearbeitung von Nachrichten verwendet, die entweder zum Einreihen oder Abrufen der Anwendung oder für den Warteschlangenmanager zu groß sind (einschließlich eingreifender Warteschlangenmanager, über die die Nachricht übermittelt wird). Weitere Informationen finden Sie unter „Nachrichtensegmentierung” auf Seite 279.

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 38 zeigt ein Beispiel für Nachrichten in einer Gruppe). Ein Segment ist logisch gesehen Teil einer einzelnen Nachricht, sodass sich im MQMD nur die Felder *MsgId*, *Offset* und *SegmentFlag* zwischen separaten Segmenten der Nachricht unterscheiden sollten. Wenn ein Segment nicht eintrifft, wird der Ursachencode MQRC_INCOMPLETE_GROUP oder MQRC_INCOMPLETE_MSG soweit erforderlich ausgegeben.

Abbildung 8 auf Seite 38 zeigt eine Gruppe logischer Nachrichten, von denen einige segmentiert sind:

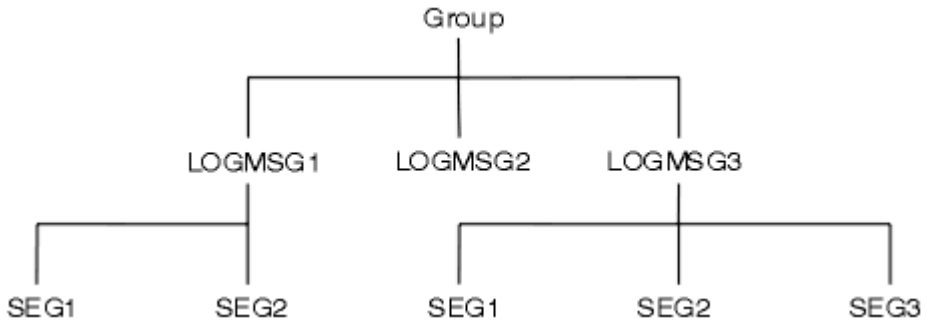


Abbildung 8. Segmentierte Nachrichten

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

Eine Beschreibung logischer und physischer Nachrichten finden Sie im Thema „Logische und physische Reihenfolge” auf Seite 261. Weitere Informationen zur Segmentierung von Nachrichten finden Sie im Thema „Nachrichtensegmentierung” auf Seite 279.

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.

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 343.
- In IBM WebSphere MQ auf UNIX -Systemen, IBM WebSphere MQ auf Linux -Systemen, und IBM WebSphere MQ für Windowswird eine persistente Nachricht in der aktuellen Arbeitseinheit nur protokolliert, wenn die Arbeitseinheit festgeschrieben wird (und die Arbeitseinheit kann viele Warteschlangenoperationen enthalten).

Nicht persistente Nachrichten können für einen schnellen Nachrichtenaustausch eingesetzt werden. Weitere Informationen zum schnellen Messaging finden Sie im Thema 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 Thema „Programmfehler behandeln” auf Seite 581.

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.

Unter WebSphere MQ for WebSphere MQ for Windows, WebSphere MQ auf UNIX-Systemen, WebSphere MQ auf Linux-Systemen bleibt der Zurücksetzungszähler beim Neustart des Warteschlangenmanagers immer erhalten. Änderungen des Attributs *HardenGetBackout* werden ignoriert.

Weitere Informationen zum Festschreiben und Rücksetzen von Nachrichten finden Sie unter „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen" auf Seite 343.

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 Zustellung nicht gelöscht, reiht sie der ferne Warteschlangenmanager in die Warteschlange für nicht zustellbare Nachrichten ein (siehe „Warteschlange für nicht zustellbare Nachrichten verwenden" auf Seite 584).

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 „Kontextinformationen steuern" auf Seite 246.

Der Benutzerkontext wird vom Warteschlangenmanager beim Erstellen folgender Berichtsnachrichtenarten verwendet:

- Empfangsbestätigung
- Verfall

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

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

Identitätskontext

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

- Der Warteschlangenmanager gibt im Feld *UserIdentifier* einen Namen an, der den Benutzer identifiziert; wie diese Identifizierung erfolgt, ist von der Umgebung abhängig, in der die Anwendung ausgeführt wird.
- Der Warteschlangenmanager trägt im Feld *AccountingToken* ein Token oder eine Zahl ein, das bzw. die aus der Anwendung ermittelt wurde, von der die Nachricht eingereiht wurde.
- Im Feld *ApplIdentityData* können Anwendungen zusätzliche Informationen zum Benutzer eintragen (beispielsweise ein verschlüsseltes Kennwort).

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

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

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

Ursprungskontext

Ursprungskontext-Informationen beschreiben die Anwendung, die die Nachricht in die Warteschlange einreiht, in der die Nachricht *aktuell* gespeichert ist. Der Nachrichtendeskriptor enthält folgende Felder für die ursprünglichen Kontextinformationen:

PutApplType	Der Typ der Anwendung, die die Nachricht einreiht (z. B. eine CICS-Transaktion).
PutApplName	Der Name der Anwendung, die die Nachricht einreiht (z. B. der Name eines Jobs oder einer Transaktion).
PutDate	Das Datum, an dem die Nachricht in die Warteschlange eingereiht wurde.
PutTime	Die Zeit, zu der die Nachricht in die Warteschlange eingereiht wurde.
ApplOriginData	Alle zusätzlichen Informationen, die eine Anwendung über den Ursprung der Nachricht einfügen möchte. 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.

WebSphere MQ-Objekte

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

Warteschlangenmanager definieren die Eigenschaften (die als 'Attribute' bezeichnet werden) dieser Objekte. Die Werte dieser Attribute bestimmen, wie diese Objekte von WebSphere MQ verarbeitet werden. In den Anwendungen werden diese Objekte über die MQI (Message Queue Interface) gesteuert. Programme verweisen auf Objekte anhand eines *Objektdeskriptors* (MQOD).

Wenn Sie mit WebSphere MQ-Befehlen Objekte definieren, ändern oder löschen, prüft der Warteschlangenmanager, ob Sie über die entsprechenden Berechtigungen für diese Aktionen 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

„Kontextinformationen steuern” auf Seite 246
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 236
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.

MTS-Anwendungen (Microsoft Transaction Server) vorbereiten und ausführen

Um eine MTS-Anwendung für ihre Ausführung als WebSphere MQ-Clientanwendung vorzubereiten, müssen Sie entsprechend den Anweisungen für Ihre Umgebung vorgehen.

Allgemeine Informationen zur Entwicklung von MTS-Awendungen (Microsoft Transaction Server), die auf WebSphere MQ-Ressourcen zugreifen, finden Sie im WebSphere MQ Help Center im Abschnitt über MTS.

Um eine MTS-Anwendung für ihre Ausführung als WebSphere MQ-Clientanwendung vorzubereiten, führen Sie für jede Komponente der Anwendung einen der folgenden Schritte aus:

- Verwendet die Komponente Bindungen der Programmiersprache C für die MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „C-Programme unter Windows vorbereiten” auf Seite 487 vorgehen, die Komponente jedoch mit der Bibliothek 'mqicxa.lib' verbinden, nicht mit 'mqic.lib'.
- Verwendet die Komponente die C++-Klassen von WebSphere MQ, müssen Sie entsprechend den Anweisungen im Thema „C++-Programme unter Windows erstellen” auf Seite 693 vorgehen, die Komponente jedoch mit der Bibliothek 'imqx23vn.lib' verbinden, 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 unter Windows vorbereiten” auf Seite 491

vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.

- Verwendet die Komponente MQAX (WebSphere MQ Automation Classes for ActiveX), müssen Sie die Umgebungsvariable GMQ_MQ_LIB mit dem Wert `mqic32xa.dll` definieren.

  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 WebSphere MQ mit WebSphere Application Server verwenden

In diesem Abschnitt wird die Verwendung von IBM WebSphere MQ zusammen mit WebSphere Application Server erläutert.

In Java erstellte Anwendungen, die unter WebSphere Application Server ausgeführt werden, können JMS-Spezifikation (Java Messaging Service) für den Nachrichtenaustausch verwenden. Punkt-zu-Punkt-Messaging in dieser Umgebung kann von einem IBM WebSphere MQ-Warteschlangenmanager bereitgestellt werden.

Die Verwendung eines IBM WebSphere MQ-Warteschlangenmanagers für das Punkt-zu-Punkt-Messaging hat den Vorteil, dass JMS-Anwendungen, die eine Verbindung herstellen, die gesamte Funktionalität eines IBM WebSphere MQ-Netzes nutzen können, die Anwendungen also Nachrichten mit Warteschlangenmanagern austauschen können, die auf vielen verschiedenen 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. Wenn der Warteschlangenmanager der Anwendung nicht lokal zur Verfügung steht, sollte der *Clientanschluss* installiert werden, damit die Anwendung eine Verbindung zu einem Warteschlangenmanager herstellen kann, der auf einer anderen Maschine oder einem anderen Image ausgeführt wird.

Standardmäßig verwenden in IBM WebSphere MQ-Warteschlangen enthaltene JMS-Nachrichten einen MQRFH2-Header zum Speichern eines Teils der JMS-Nachrichtenheaderinformationen. Viele traditionelle IBM WebSphere MQ-Anwendungen können Nachrichten, die solche Header enthalten, nicht verarbeiten; sie benötigen eigene charakteristische Header - CICS-Bridge-Anwendungen benötigen beispielsweise einen MQCIH-Header, IBM WebSphere MQ-Workflowanwendungen einen MQWIH-Header. Weitere Details zu diesen besonderen Aspekten finden Sie unter „Zuordnung von JMS-Nachrichten zu WebSphere MQ-Nachrichten” auf Seite 863.

Transaktionsunterstützungsszenarios

Mittels Transaktionsunterstützung konfigurieren Sie für Ihre Anwendungen eine zuverlässige Interaktion mit Datenbanken.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

In diesem Abschnitt wird die Transaktionsunterstützung beschrieben. Damit Ihre Anwendungen IBM WebSphere MQ mit einem Datenbankprodukt verwenden können, sind u. a. Maßnahmen im Bereich der Anwendungsprogrammierung und Systemverwaltung erforderlich. Verwenden Sie die hier aufgeführten Informationen zusammen mit den Angaben im Thema „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343.

Zunächst sollen die einzelnen Arbeitseinheiten der Transaktionen vorgestellt werden, anschließend wird beschrieben, wie IBM WebSphere MQ für die Koordination von Transaktionen mit Datenbanken vorbereitet werden kann.

Zugehörige Konzepte

„Arbeitseinheiten - Einführung” auf Seite 44

In diesem Thema werden die allgemeinen Konzepte 'Arbeitseinheit', 'Festschreibung', 'Backout' und 'Synchronisationspunkt' vorgestellt und definiert. Außerdem finden Sie hier zwei Szenarios zur Veranschaulichung globaler Arbeitseinheiten.

IBM WebSphere MQ und HP NonStop TMF

Arbeitseinheiten - Einführung

In diesem Thema werden die allgemeinen Konzepte 'Arbeitseinheit', 'Festschreibung', 'Backout' und 'Synchronisationspunkt' vorgestellt und definiert. Außerdem finden Sie hier zwei Szenarios zur Veranschaulichung globaler Arbeitseinheiten.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wenn ein Programm Nachrichten in Warteschlangen innerhalb einer Arbeitseinheit einreiht, werden diese Nachrichten für andere Programme erst sichtbar, wenn das Programm die Arbeitseinheit *festschreibt*. Zur Gewährleistung der Datenintegrität müssen für das Commit einer Arbeitseinheit alle Aktualisierungen erfolgreich durchgeführt worden sein.

Falls das Programm einen Fehler erkennt und beschließt, die Put-Operation nicht permanent zu machen, kann es die Arbeitseinheit zurücksetzen (*Backout*). Wenn ein Programm ein Backout durchführt, stellt WebSphere MQ die Warteschlangen wieder her und entfernt zu diesem Zweck die Nachrichten, die von der betreffenden Arbeitseinheit in die Warteschlangen eingereiht wurden.

Ähnlich gilt: Wenn ein Programm Nachrichten aus einer oder mehreren Warteschlangen innerhalb einer Arbeitseinheit abruft, bleiben diese Nachrichten in den Warteschlangen, bis das Programm die Arbeitseinheit festschreibt, können jedoch nicht von anderen Programmen abgerufen werden. Wenn das Programm die Arbeitseinheit festschreibt, werden die Nachrichten permanent aus den Warteschlangen gelöscht. Bei einem Backout der Arbeitseinheit durch das Programm stellt WebSphere MQ die Warteschlangen wieder her, indem es sie wieder für den Abruf durch andere Programme zur Verfügung stellt.

Die Entscheidung über Festschreibung oder Backout der Änderungen fällt im einfachsten Fall am Ende einer Task. Es kann jedoch für eine Anwendung sinnvoller sein, Datenänderungen an anderen logischen Punkten innerhalb einer Task zu synchronisieren. Diese logischen Punkte werden Synchronisationspunkte genannt, der Zeitraum der Verarbeitung einer Reihe von Aktualisierungen zwischen zwei Synchronisationspunkten wird als *Arbeitseinheit* bezeichnet. Eine einzelne Arbeitseinheit kann mehrere MQGET- und MQPUT-Aufrufe umfassen.

Bei WebSphere MQ muss zwischen *lokalen* und *globalen* Arbeitseinheiten unterschieden werden:

Lokale Arbeitseinheiten
Arbeitseinheiten, bei denen lediglich Nachrichten in WebSphere MQ-Warteschlangen eingereiht und daraus abgerufen werden und die Koordination der einzelnen Arbeitseinheiten innerhalb des Warteschlangenmanagers mithilfe einer *einphasigen Festschreibung* erfolgt.

Verwenden Sie lokale Arbeitseinheiten, wenn als einzige Ressourcen Warteschlangen aktualisiert werden müssen, die von einem einzelnen WebSphere MQ-Warteschlangenmanager verwaltet werden. Aktualisierungen werden mit dem Verb MQCMIT festgeschrieben oder mit MQBACK zurückgesetzt.

Im Zusammenhang mit lokalen Arbeitseinheiten fallen abgesehen von der Protokollverwaltung keine Systemverwaltungstasks an. Versuchen Sie bei Anwendungen, bei denen Sie die Aufrufe MQPUT und MQGET mit MQCMIT und MQBACK verwenden, die Optionen MQPMO_SYNCPOINT und MQGMO_SYNCPOINT zu verwenden. (Informationen zur Protokollverwaltung finden Sie im Thema Protokolldateien verwalten.)

Globale Arbeitseinheiten
Arbeitseinheiten, in denen auch andere Ressourcen wie z. B. Tabellen in einer relationalen Datenbank aktualisiert werden. Ist mehr als ein *Ressourcenmanager* beteiligt, wird *Transaktionsmanager*-Software benötigt, die eine *zweiphasige Festschreibung* zur Koordination der globalen Arbeitseinheit verwendet.

Verwenden Sie globale Arbeitseinheiten, wenn auch Aktualisierungen der Manager-Software für relationale Datenbanken (Db2, Oracle, Sybase oder Informix) berücksichtigt werden müssen.

Es gibt mehrere mögliche Szenarios für die Verwendung globaler Arbeitseinheiten. Hier sind zwei Szenarios dokumentiert:

1. Im ersten Szenario agiert der Warteschlangenmanager selbst als Transaktionsmanager. In diesem Szenario werden die globalen Arbeitseinheiten über MQI-Verben gesteuert; mit dem Verb MQBEGIN werden sie in Anwendungen gestartet und anschließend mit MQCMIT festgeschrieben oder mit MQBACK zurückgesetzt.
2. Im zweiten Szenario ist für die Rolle des Transaktionsmanagers andere Software, beispielsweise TXSeries, Encina oder Tuxedo, zuständig. In diesem Szenario wird die Arbeitseinheit über eine von der Transaktionsmanagersoftware bereitgestellte API gesteuert (z. B. CICS SYNCPOINT für TXSeries).

In den folgenden Abschnitten werden - für das jeweilige Szenario - alle zur Verwendung globaler Arbeitseinheiten erforderlichen Schritte beschrieben:

- „Szenario 1: Der Warteschlangenmanager nimmt die Koordinierung vor” auf Seite 45
- „Szenario 2: Für die Koordination ist andere Software zuständig” auf Seite 72

Szenario 1: Der Warteschlangenmanager nimmt die Koordinierung vor

In Szenario 1 agiert der Warteschlangenmanager als Transaktionsmanager. In diesem Szenario werden die globalen Arbeitseinheiten über MQI-Verben gesteuert; mit dem Verb MQBEGIN werden sie in Anwendungen gestartet und anschließend mit MQCMIT festgeschrieben oder mit MQBACK zurückgesetzt.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Isolationsstufe

In IBM WebSphere MQ kann es je nach in der Datenbank implementierter Transaktionsisolationsgestaltung vorkommen, dass vor einem Datenbankupdate eine Nachricht in einer Warteschlange zu sehen ist.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wenn ein IBM WebSphere MQ-Warteschlangenmanager als XA-Transaktionsmanager fungiert, wird zur Koordination von Aktualisierungen der XA-Ressourcenmanager das folgende Festschreibungsprotokoll befolgt:

1. Vorbereitung aller XA-Ressourcenmanager.
2. Schreiben Sie den Warteschlangenmanager IBM WebSphere MQ fest.
3. Festschreibung anderer Ressourcenmanager.

Zwischen Schritt 2 und Schritt 3 kann es vorkommen, dass eine Anwendung eine in der Warteschlange festgeschriebene Nachricht sieht, diese jedoch in der entsprechenden Zeile in der Datenbank nicht wiedergegeben ist.

Dies ist kein Problem, wenn die Datenbank so konfiguriert ist, dass die Datenbank-API-Aufrufe der Anwendung auf den Abschluss ausstehender Aktualisierungen warten.

Sie können diese Situation beheben, indem Sie die Datenbank anders konfigurieren. Der benötigte Konfigurationstyp wird als "Isolationsstufe" bezeichnet. Weitere Informationen zu Isolationsstufen finden Sie in der Datenbankdokumentation. Alternativ dazu können Sie den Warteschlangenmanager so konfigurieren, dass die Ressourcenmanager in der folgenden umgekehrten Reihenfolge festgeschrieben werden.

1. Vorbereitung aller XA-Ressourcenmanager.
2. Festschreibung anderer Ressourcenmanager.
3. Schreiben Sie den Warteschlangenmanager IBM WebSphere MQ fest.

Bei einer Protokolländerung wird der IBM WebSphere MQ-Warteschlangenmanager zuletzt festgeschrieben, daher erhalten Anwendungen, die Nachrichten aus den Warteschlangen lesen, erst nach erfolgter Aktualisierung der entsprechenden Datenbank eine Nachricht.

Definieren Sie zur Konfiguration des Warteschlangenmanagers für die Verwendung dieses geänderten Protokolls die Umgebungsvariable **AMQ_REVERSE_COMMIT_ORDER**.

Definieren Sie diese Umgebungsvariable in der Umgebung, in welcher der Befehl **strmqm** zum Starten des Warteschlangenmanagers ausgeführt wird. Führen Sie beispielsweise unmittelbar vor dem Start des Warteschlangenmanagers den folgenden Befehl in der Shell aus:

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

Anmerkung: Wenn Sie diese Umgebungsvariable definieren, wird möglicherweise für jede Transaktion ein zusätzlicher Protokolleintrag erstellt, dies hat also einen leichten Einfluss auf die Leistung der einzelnen Transaktionen.

Datenbankkordination

Wenn der Warteschlangenmanager die globalen Arbeitseinheiten selbst koordiniert, ist es möglich, Datenbankaktualisierungen in die Arbeitseinheiten zu integrieren. Somit kann eine heterogene MQI- und SQL-Anwendung geschrieben werden und mithilfe der Verben MQCMIT und MQBACK können die Änderungen an den Warteschlangen und Datenbanken zusammen festgeschrieben oder rückgängig gemacht werden.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Der Warteschlangenmanager verwendet hierfür das in *X/Open Distributed Transaction Processing: The XA Specification* beschriebene Protokoll für zweiphasige Festschreibung. Wenn eine Arbeitseinheit festgeschrieben werden muss, fragt der Warteschlangenmanager zunächst alle beteiligten Datenbankmanager, ob sie zur Festschreibung ihrer Aktualisierungen bereit sind. Nur wenn alle Beteiligten einschließlich des Warteschlangenmanagers selbst zur Festschreibung bereit sind, werden alle Aktualisierungen der Warteschlangen und Datenbanken festgeschrieben. Falls ein Beteiligter seine Aktualisierungen nicht vorbereiten kann, wird die Arbeitseinheit stattdessen zurückgesetzt.

Im Allgemeinen wird eine globale Arbeitseinheit mit der folgenden Methode in einer Anwendung implementiert (in Pseudocode):

```
MQBEGIN
MQGET (nehmen Sie das Flag MQGMO_SYNCPOINT in die Nachrichtenoptionen auf)
MQPUT (nehmen Sie das Flag MQPMO_SYNCPOINT in die Nachrichtenoptionen auf)
SQL INSERT
MQCMIT
```

Mit MQBEGIN wird der Anfang einer globalen Arbeitseinheit bezeichnet. Mit MQCMIT wird das Ende der globalen Arbeitseinheit bezeichnet und die Arbeitseinheit mit allen beteiligten Ressourcenmanagern unter Verwendung des Protokolls für zweiphasige Festschreibung abgeschlossen.

Nach erfolgreichem Abschluss der Arbeitseinheit (auch als *Transaktion* bezeichnet) mit MQCMIT sind alle Aktionen innerhalb der Arbeitseinheit permanent bzw. irreversibel. Falls die Arbeitseinheit aus irgendeinem Grund fehlschlägt, werden alle Aktionen stattdessen zurückgesetzt. Es ist nicht möglich, eine Aktion in einer Arbeitseinheit permanent zu machen, während eine andere zurückgesetzt wird. Darin besteht das Prinzip einer Arbeitseinheit: entweder werden alle Aktionen innerhalb der Arbeitseinheit permanent gemacht oder keine.

Anmerkung:

1. Durch Aufruf von MQBACK kann der Anwendungsprogrammierer die Zurücksetzung (Backout) einer Arbeitseinheit erzwingen. Außerdem wird die Arbeitseinheit vom Warteschlangenmanager zurückgesetzt, wenn die Anwendung oder Datenbank vor Aufruf von MQCMIT *fehlschlägt*.

2. Wenn eine Anwendung MQDISC, nicht jedoch MQCMIT aufruft, verfährt der Warteschlangenmanager, als ob MQCMIT aufgerufen worden wäre, und schreibt die Arbeitseinheit fest.

Zwischen MQBEGIN und MQCMIT richtet der Warteschlangenmanager keine Aufrufe an die Datenbank zur Ressourcenaktualisierung. Somit werden die Datenbanktabellen also nur durch Ihren Code geändert (beispielsweise durch SQL INSERT im Pseudocode).

Wenn der Warteschlangenmanager während des Festschreibungsprotokolls den Kontakt zu einem der Datenbankmanager verliert, wird Gesamtwiederherstellungsunterstützung geboten. Wenn ein unbestätigter Datenbankmanager nicht mehr verfügbar ist, also ein Datenbankmanager, der erfolgreich für die Festschreibung vorbereitet wurde, jedoch noch auf eine Festschreibungs- oder Backoutentscheidung wartet, merkt sich der Warteschlangenmanager das Ergebnis der Arbeitseinheit, bis das betreffende Ergebnis erfolgreich an die Datenbank übergeben wurde. Ähnlich gilt: wenn der Warteschlangenmanager mit ausstehenden unvollständigen Festschreibungsoperationen beendet wird, bleiben diese über einen Neustart des Warteschlangenmanagers hinweg registriert. Falls eine Anwendung unerwartet beendet wird, ist die Integrität der Arbeitseinheit nicht beeinträchtigt, das Ergebnis hängt allerdings davon ab, an welcher Stelle des Prozesses die Anwendung beendet wurde (siehe Tabelle 5 auf Seite 47).

In den folgenden Tabellen ist zusammengefasst dargestellt, was geschieht, wenn die Datenbank oder das Anwendungsprogramm fehlschlägt:

Tabelle 4. Was geschieht, wenn ein Datenbankserver fehlschlägt?

Auftreten des Fehlers	**Ergebnis**
Vor Anwendungsaufruf von MQCMIT.	Die Arbeitseinheit wird zurückgesetzt.
Während des Anwendungsaufrufs von MQCMIT, **bevor** alle Datenbanken angegeben haben, dass sie erfolgreich Vorbereitungen getroffen haben.	Die Arbeitseinheit wird mit dem Ursachencode MQRC_BACKED_OUT zurückgesetzt.
Während des Anwendungsaufrufs von MQCMIT, **nachdem** alle Datenbanken angegeben haben, dass sie erfolgreich Vorbereitungen getroffen haben, jedoch bevor alle die erfolgreiche Festschreibung gemeldet haben.	Die Arbeitseinheit wird vom Warteschlangenmanager in wiederherstellbarem Status gehalten, mit dem Ursachencode MQRC_OUTCOME_PENDING.
Während des Anwendungsaufrufs von MQCMIT, **nachdem** alle Datenbanken die erfolgreiche Festschreibung gemeldet haben.	Die Arbeitseinheit wird mit dem Ursachencode MQRC_NONE festgeschrieben.
Nach dem Anwendungsaufruf von MQCMIT.	Die Arbeitseinheit wird mit dem Ursachencode MQRC_NONE festgeschrieben.

Tabelle 5. Was geschieht, wenn ein Anwendungsprogramm fehlschlägt?

Auftreten des Fehlers	**Ergebnis**
Vor Anwendungsaufruf von MQCMIT.	Die Arbeitseinheit wird zurückgesetzt.
Während des Anwendungsaufrufs von MQCMIT, **bevor** der Warteschlangenmanager die Anforderung MQCMIT der Anwendung erhalten hat.	Die Arbeitseinheit wird zurückgesetzt.
Während des Anwendungsaufrufs von MQCMIT, **nachdem** der Warteschlangenmanager die Anforderung MQCMIT der Anwendung erhalten hat.	Der Warteschlangenmanager versucht eine zweiphasige Festschreibung (vorausgesetzt, die Datenbankprodukte führen ihre Teile der Arbeitseinheit erfolgreich aus und schreiben diese fest).

Falls von MQCMIT der Ursachencode MQRC_OUTCOME_PENDING zurückgegeben wird, merkt sich der Warteschlangenmanager die Arbeitseinheit, bis er wieder Kontakt zum Datenbankserver herstellen und diesen auffordern kann, seinen Teil der Arbeitseinheit festzuschreiben. Informationen zur Vorgehenswei-

se sowie zum Zeitpunkt der Wiederherstellung finden Sie im Abschnitt „Überlegungen zur Vorgehensweise bei Verlust des Kontakts zum XA-Ressourcenmanager” auf Seite 65.

Der Warteschlangenmanager kommuniziert über die XA-Schnittstelle mit den Datenbankmanagern, wie in *X/Open Distributed Transaction Processing: The XA Specification* beschrieben. Beispiele für diese Funktionsaufrufe sind 'xa_open', 'xa_start', 'xa_end', 'xa_prepare' und 'xa_commit'. Wir verwenden die Begriffe *Transaktionsmanager* und *Ressourcenmanager* im Sinne der XA-Spezifikation.

Einschränkungen
Bezüglich der Unterstützung der Datenbankkoordination gibt es gewisse Einschränkungen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Es gelten folgende Einschränkungen:

- Die Möglichkeit der Koordination von Datenbankaktualisierungen innerhalb von WebSphere MQ-Arbeitseinheiten wird in einer MQI-Clientanwendung **nicht** unterstützt. Bei Verwendung von MQBEGIN in einer Clientanwendung kommt es zu einem Fehler. Ein Programm, das MQBEGIN aufruft, muss als *Serveranwendung* auf demselben System wie der Warteschlangenmanager ausgeführt werden.

  Anmerkung: Bei einer *Serveranwendung* handelt es sich um ein Programm, das mit den erforderlichen WebSphere MQ-Serverbibliotheken verknüpft wurde; eine *Clientanwendung* ist ein Programm, das mit den erforderlichen WebSphere MQ-Clientbibliotheken verknüpft wurde. Weitere Informationen zum Kompilieren und Verknüpfen Ihrer Programme finden Sie in den Themen „Erstellen von Anwendungen für WebSphere MQ-Clients” auf Seite 380 und „IBM WebSphere MQ-Anwendung erstellen” auf Seite 456.
- Der Datenbankserver kann sich auf einem anderen System als der Warteschlangenmanagerserver befinden, wenn der Datenbankclient auf demselben System wie der Warteschlangenmanager installiert ist und diese Funktion unterstützt. Prüfen Sie in der Dokumentation zu dem Datenbankprodukt, ob die Client-Software für Systeme mit zweiphasiger Festschreibung verwendet werden kann.
- Zwar fungiert der Warteschlangenmanager als Ressourcenmanager (um in die globalen Arbeitseinheiten aus Szenario 2 einbezogen zu werden), es ist jedoch nicht möglich, einen Warteschlangenmanager einen anderen Warteschlangenmanager innerhalb der globalen Arbeitseinheiten von Szenario 1 koordinieren zu lassen.

Switchloaddateien

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Bei der Switchloaddatei handelt es sich um eine freigegebene Bibliothek (eine DLL in Windows-Systemen), die vom Code in Ihrer IBM WebSphere MQ-Anwendung und dem Warteschlangenmanager geladen wird. Mit dieser Datei soll das Laden der freigegebenen Clientbibliothek der Datenbank vereinfacht werden, außerdem sollen die Verweise auf die XA-Funktionen zurückgegeben werden.

Die Details der Switchloaddatei müssen vor dem Start des Warteschlangenmanagers angegeben werden. Die Angaben werden in die Datei 'qm.ini' auf Windows-, UNIX and Linux-Systemen gestellt.

- Verwenden Sie unter Windows und Linux (x86- und x86-64-Plattformen) IBM WebSphere MQ Explorer zum Aktualisieren der Datei 'qm.ini'.
- Bearbeiten Sie die Datei 'qm.ini' auf allen anderen Systemen direkt.

Die C-Quelle für die Switchloaddatei wird mit der IBM WebSphere MQ-Installation bereitgestellt, falls die globalen Arbeitseinheiten aus Szenario 1 unterstützt werden. Die Quelle enthält eine Funktion mit der Bezeichnung 'MQStart'. Beim Laden der Switchloaddatei ruft der Warteschlangenmanager diese Funktion auf, die die Adresse einer Struktur, eines sogenannten *XA-Switches*, zurückgibt.

Die XA-Switchstruktur ist in der freigegebenen Datenbankclientbibliothek enthalten und umfasst eine Reihe von Funktionszeigern, wie in Tabelle 6 auf Seite 49 beschrieben:

Tabelle 6. XA-Switch-Funktionszeiger		
Funktionszeigername	**XA-Funktion**	**Verwendungszweck**
xa_open_entry	xa_open	Verbindung zur Datenbank
xa_close_entry	xa_close	Trennung der Verbindung zur Datenbank
xa_start_entry	xa_start	Starten einer Verzweigung einer globalen Arbeitseinheit
xa_end_entry	xa_end	Aussetzen einer Verzweigung einer globalen Arbeitseinheit
xa_rollback_entry	xa_rollback	Rollback einer Verzweigung einer globalen Arbeitseinheit
xa_prepare_entry	xa_prepare	Vorbereitung der Festschreibung einer Verzweigung einer globalen Arbeitseinheit
xa_commit_entry	xa_commit	Festschreibung einer Verzweigung einer globalen Arbeitseinheit
xa_recover_entry	xa_recover	Erkennen unbestätigter Arbeitseinheiten in der Datenbank
xa_forget_entry	xa_forget	Übergehen einer Verzweigung einer globalen Arbeitseinheit durch die Datenbank zulassen
xa_complete_entry	xa_complete	Beendigung einer Verzweigung einer globalen Arbeitseinheit

Während des ersten Aufrufs MQBEGIN in Ihrer Anwendung lädt der im Rahmen von MQBEGIN ausgeführte IBM WebSphere MQ-Code die Switchloaddatei und ruft die Funktion 'xa_open' in der freigegebenen Datenbankbibliothek auf. In ähnlicher Weise laden manche Warteschlangenmanagerprozesse beim Start des Warteschlangenmanagers und späteren Gelegenheiten die Switchloaddatei und rufen die Funktion 'xa_open' auf.

Über die *dynamische Registrierung* kann die Zahl der Aufrufe 'xa_*' reduziert werden. Eine umfassende Beschreibung dieses Optimierungsverfahrens finden Sie im Abschnitt „Dynamische XA-Registrierung” auf Seite 69.

System für die Datenbankkoordination konfigurieren

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Bevor ein Datenbankmanager an den vom Warteschlangenmanager koordinierten globalen Arbeitseinheiten teilnehmen kann, müssen mehrere Tasks ausgeführt werden. Diese werden in den folgenden Abschnitten beschrieben:

- „Installation und Konfiguration des Datenbankprodukts” auf Seite 50
- „Switchloaddateien erstellen” auf Seite 50
- „Konfigurationsinformationen zum Warteschlangenmanager hinzufügen” auf Seite 51
- „Anwendungen schreiben und ändern” auf Seite 53
- „System testen” auf Seite 53

Installation und Konfiguration des Datenbankprodukts
Beachten Sie für die Installation und Konfiguration Ihres Datenbankprodukts die produkteigene Dokumentation. In den Themen in diesem Abschnitt werden allgemeine Aspekte im Zusammenhang mit der Konfiguration beschrieben und es wird erläutert, wie sich diese auf den gemeinsamen Einsatz von WebSphere MQ und der Datenbank auswirken.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Datenbankverbindungen

Anwendungen, die eine Standardverbindung zum Warteschlangenmanager aufbauen, werden einem Thread in einem separaten Agentenprozess des lokalen Warteschlangenmanagers zugeordnet. (In diesem Zusammenhang gelten Verbindungen, bei denen es sich nicht um *Fastpath*-Verbindungen handelt, als *Standardverbindungen*. Weitere Informationen finden Sie im Thema „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 222.)

Wenn die Anwendung `MQBEGIN` ausgibt, wird die Funktion 'xa_open' in der Datenbankclientbibliothek sowohl von der Anwendung selbst als auch vom Agentenprozess aufgerufen. Als Antwort darauf stellt der Code der Datenbankclientbibliothek *sowohl von den Anwendungs- als auch von den Warteschlangenmanagerprozessen* aus eine *Verbindung* zu der Datenbank her, die in die Arbeitseinheit einbezogen werden soll. Diese Datenbankverbindungen bleiben erhalten, solange die Anwendung mit dem Warteschlangenmanager verbunden bleibt.

Dies ist ein wichtiger Gesichtspunkt, wenn die Datenbank nur eine begrenzte Anzahl an Benutzern oder Verbindungen unterstützt, da zur Unterstützung des einen Anwendungsprogramms zwei Verbindungen zur Datenbank hergestellt werden.

Client-/Serverkonfiguration

Die in die WebSphere MQ-Warteschlangenmanager- und -Anwendungsprozesse geladene Datenbankclientbibliothek **muss** in der Lage sein, Daten an den Server zu senden und von dort zu empfangen. Vergewissern Sie sich, dass folgende Voraussetzungen erfüllt sind:

- Die Angaben in den Client-/Server-Konfigurationsdateien der Datenbank sind korrekt.
- In der Umgebung der Warteschlangenmanager- **und** Anwendungsprozesse die relevanten Umgebungsvariablen definiert.

Switchloaddateien erstellen
WebSphere MQ wird mit einer Beispielmakefile geliefert, die zum Erstellen von Switchloaddateien für die unterstützten Datenbankmanager verwendet wird.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

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

Die Beispielmakedatei wird zusammen mit allen zum Erstellen der Switchloaddateien erforderlichen zugehörigen C-Quellendateien in den folgenden Verzeichnissen installiert:

- Für WebSphere MQ für Windowsim Verzeichnis *`MQ_INSTALLATION_PATH`*`\tools\c\samples\xatm\`
- Bei WebSphere MQ für UNIX and Linux -Systeme im Verzeichnis *`MQ_INSTALLATION_PATH`*`/samp/xatm/`

Zum Erstellen der Switchloaddateien werden die folgenden Beispielquellenmodule verwendet:

- db2swit.c für DB2
- oraswit.c für Oracle
- infswit.c für Informix

- sybswit.c für Sybase

Installieren Sie beim Erstellen von Switchloaddateien die 32-Bit-Switchloaddateien im Verzeichnis `/var/mqm/exits` und die 64-Bit-Switchloaddateien im Verzeichnis `/var/mqm/exits64`.

Bei Verwendung von 32-Bit-Warteschlangenmanagern installiert die Beispielmakedatei 'xaswit.mak' eine 32-Bit-Switchloaddatei im Verzeichnis `/var/mqm/exits`.

Bei Verwendung von 64-Bit-Warteschlangenmanagern installiert die Beispielmakedatei 'xaswit.mak' eine 32-Bit-Switchloaddatei im Verzeichnis `/var/mqm/exits` und eine 64-Bit-Switchloaddatei im Verzeichnis `/var/mqm/exits64`.

Dateisicherheit

Möglicherweise kann die Switchloaddatei von WebSphere MQ unter Ihrem Betriebssystem nicht geladen werden, wobei die Gründe hierfür außerhalb der Kontrolle von WebSphere MQ liegen. In diesem Fall werden Fehlernachrichten in die WebSphere MQ-Fehlerprotokolle geschrieben und unter Umständen schlägt der Aufruf MQBEGIN fehl. Um sicherzugehen, dass die Switchloaddatei unter Ihrem Betriebssystem geladen werden kann, müssen folgende Voraussetzungen erfüllt sein:

1. Die Switchloaddatei muss sich in dem in der Datei 'qm.ini' angegebenen Verzeichnis befinden.
2. Die Switchloaddatei muss für alle Prozesse zugänglich sein, die diese laden müssen, einschließlich der Warteschlangenmanager- und Anwendungsprozesse.
3. Alle Bibliotheken, von denen die Switchloaddatei abhängt, einschließlich der vom Datenbankprodukt bereitgestellten Bibliotheken, müssen vorhanden und zugänglich sein.

Konfigurationsinformationen zum Warteschlangenmanager hinzufügen
Wenn Sie eine Switchloaddatei für Ihren Datenbankmanager erstellt und in ein sicheres Verzeichnis gestellt haben, müssen Sie dieses Verzeichnis Ihrem Warteschlangenmanager angeben.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Gehen Sie zur Angabe des Verzeichnisses wie folgt vor:

- Auf Windows- und Linux -Systemen (x86 -und x86-64 -Plattformen) verwenden Sie den WebSphere MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an.
- Geben Sie bei allen anderen Systemen die Details zu der Switchloaddatei in der XAResourceManager-Zeilengruppe in der Datei 'qm.ini' des Warteschlangenmanagers an.

Fügen Sie für die Datenbank, die Ihr Warteschlangenmanager koordinieren soll, eine XAResourceManager-Zeilengruppe hinzu. In den meisten Fällen gibt es nur eine Datenbank und daher auch nur eine XAResourceManager-Zeilengruppe. Ausführliche Informationen zu komplizierteren Konfigurationen mit mehreren Datenbanken finden Sie im Abschnitt „Mehrere Datenbankkonfigurationen” auf Seite 63. Die Attribute der XAResourceManager-Zeilengruppe lauten wie folgt:

Name=name
Vom Benutzer ausgewählte Zeichenfolge zur Angabe des Ressourcenmanagers. Damit wird der XAResourceManager-Zeilengruppe praktisch ein Name gegeben. Der Name ist obligatorisch und kann bis zu 31 Zeichen umfassen.

Sie müssen einen eindeutigen Namen auswählen; es darf nur eine XAResourceManager-Zeilengruppe mit diesem Namen in der Datei 'qm.ini' geben. Der Name sollte auch aussagekräftig sein, da der Warteschlangenmanager diesen Namen sowohl in den Fehlerprotokollnachrichten des Warteschlangenmanagers als auch in der Ausgabe des Befehls `dspmqtrn` zur Angabe dieses Ressourcenmanagers verwendet. (Weitere Informationen hierzu finden Sie im Abschnitt „Ausstehende Arbeitseinheiten mit dem Befehl 'dspmqtrn' anzeigen” auf Seite 66.)

Ändern Sie das Attribut 'Name' nicht mehr, nachdem Sie einen Namen ausgewählt und den Warteschlangenmanager gestartet haben. Weitere Informationen zum Ändern der Konfigurationsinformationen finden Sie im Abschnitt „Konfigurationsinformationen ändern” auf Seite 68.

SwitchFile=name
Dies ist der Name der zuvor erstellten XA-Switchloaddatei. Hierbei handelt es sich um ein obligatorisches Attribut. Der Code in den Warteschlangenmanager- und WebSphere MQ-Anwendungsprozessen versucht bei zwei Gelegenheiten, die Switchloaddatei zu laden:

1. Beim Start des Warteschlangenmanagers
2. Beim ersten Aufruf von MQBEGIN in Ihrem WebSphere MQ-Anwendungsprozess

Die Prozesse müssen über die Sicherheits- und Berechtigungsattribute der Switchloaddatei zu dieser Aktion berechtigt sein.

XAOpenString=string
Eine Datenzeichenfolge, die der WebSphere MQ-Code in seinen Aufrufen an die Funktion 'xa_open' des Datenbankmanagers übergibt. Hierbei handelt es sich um ein optionales Attribut. Wenn es nicht angegeben ist, wird eine Zeichenfolge der Länge null vorausgesetzt.

Der Code in den Warteschlangenmanager- und WebSphere MQ-Anwendungsprozessen ruft die Funktion 'xa_open' bei zwei Gelegenheiten auf:

1. Beim Start des Warteschlangenmanagers
2. Beim ersten Aufruf von MQBEGIN in Ihrem WebSphere MQ-Anwendungsprozess

Für diese Zeichenfolge gilt bei jedem Datenbankprodukt ein besonderes Format, das in der zugehörigen Produktdokumentation beschrieben wird. Im Allgemeinen enthält die Zeichenfolge 'xa_open' Authentifizierungsinformationen (Benutzername und Kennwort), mit denen in Warteschlangenmanager- und Anwendungsprozessen eine Verbindung zu der Datenbank hergestellt werden kann.

XACloseString=string
Eine Datenzeichenfolge, die der WebSphere MQ-Code in seinen Aufrufen der Datenbankmanagerfunktion 'xa_close' übergibt. Hierbei handelt es sich um ein optionales Attribut. Wenn es nicht angegeben ist, wird eine Zeichenfolge der Länge null vorausgesetzt.

Der Code in den Warteschlangenmanager- und WebSphere MQ-Anwendungsprozessen ruft die Funktion 'xa_close' bei zwei Gelegenheiten auf:

1. Beim Start des Warteschlangenmanagers
2. Beim Aufruf von MQDISC in Ihrem WebSphere MQ-Anwendungsprozess nach einem vorherigen Aufruf von MQBEGIN

Für diese Zeichenfolge gilt bei jedem Datenbankprodukt ein besonderes Format, das in der zugehörigen Produktdokumentation beschrieben wird. Im Allgemeinen ist ist die Zeichenfolge leer, das Attribut 'XACloseString' wird häufig aus der XAResourceManager-Zeilengruppe ausgeschlossen.

ThreadOfControl=THREAD|PROCESS
Der Wert für 'ThreadOfControl' kann THREAD oder PROCESS lauten. Der Warteschlangenmanager verwendet das Attribut für Serialisierungszwecke. Hierbei handelt es sich um ein optionales Attribut, wenn es nicht angegeben ist, wird der Wert PROCESS vorausgesetzt.

Wenn der Code für den Datenbankclient den Threads den Aufruf der XA-Funktionen ohne Serialisierung ermöglicht, kann der Wert für 'ThreadOfControl' THREAD lauten. Der Warteschlangenmanager geht davon aus, dass er die XA-Funktionen in der freigegebenen Bibliothek des Datenbankclients gegebenenfalls von mehreren Threads aus gleichzeitig aufrufen kann.

Lässt der Code für den Datenbankclient nicht zu, dass Threads die XA-Funktionen auf diese Weise aufrufen, muss der Wert für 'ThreadOfControl' PROCESS lauten. Der Warteschlangenmanager serialisiert dann alle Aufrufe der freigegebenen Datenbankclientbibliothek, sodass immer nur jeweils ein Aufruf aus einem bestimmten Prozess erfolgt. Vermutlich müssen Sie auch dafür sorgen, dass Ihre Anwendung eine ähnliche Serialisierung vornimmt, wenn sie mit mehreren Threads ausgeführt wird.

Beachten Sie, dass die Fähigkeit des Datenbankprodukts, mit Multithread-Prozessen auf diese Weise zu verfahren, Sache des betreffenden Produktanbieters ist. Lesen Sie in der Dokumentation zu Ihrem

Datenbankprodukt nach, ob Sie das Attribut 'ThreadOfControl' auf THREAD oder PROCESS setzen können. Wir empfehlen, falls möglich die Einstellung THREAD für 'ThreadOfControl' zu verwenden. Im Zweifelsfall ist es *sicherer*, die Einstellung PROCESS zu verwenden, Sie verlieren dann allerdings die potenziellen Leistungsvorteile der Verwendung von THREAD.

Anwendungen schreiben und ändern
Hier wird die Vorgehensweise zum Implementieren einer globalen Arbeitseinheit erläutert.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Die mit WebSphere MQ-Installationen bereitgestellten Beispielanwendungsprogramme für die globalen Arbeitseinheiten aus Szenario 1 werden im Thema „Arbeitseinheiten - Einführung” auf Seite 44 beschrieben.

Im Allgemeinen wird eine globale Arbeitseinheit mit der folgenden Methode in einer Anwendung implementiert (in Pseudocode):

```
MQBEGIN
MQGET
MQPUT
SQL INSERT
MQCMIT
```

Mit MQBEGIN wird der Anfang einer globalen Arbeitseinheit bezeichnet. Mit MQCMIT wird das Ende der globalen Arbeitseinheit bezeichnet und die Arbeitseinheit mit allen beteiligten Ressourcenmanagern unter Verwendung des Protokolls für zweiphasige Festschreibung abgeschlossen.

Zwischen MQBEGIN und MQCMIT richtet der Warteschlangenmanager keine Aufrufe an die Datenbank zur Ressourcenaktualisierung. Somit werden die Datenbanktabellen also nur durch Ihren Code geändert (beispielsweise durch SQL INSERT im Pseudocode).

Die Rolle des Warteschlangenmanagers in Hinblick auf die Datenbank besteht darin, dieser mitzuteilen, wenn eine globale Arbeitseinheit gestartet oder beendet wurde und ob die globale Arbeitseinheit festgeschrieben oder rückgängig gemacht werden sollte.

Was Ihre Anwendung betrifft, übt der Warteschlangenmanager zwei Rollen aus: er ist Ressourcenmanager (wenn die Ressourcen Nachrichten in Warteschlangen sind) und Transaktionsmanager für die globale Arbeitseinheit.

Beginnen Sie mit den bereitgestellten Beispielprogrammen und arbeiten Sie die verschiedenen WebSphere MQ- und Datenbank-API-Aufrufe durch, die in diesen Programmen erfolgen. Die jeweiligen API-Aufrufe sind vollständig in den Themen „WebSphere MQ-Beispielprogramme” auf Seite 100 und In der MQI verwendete Datentypen sowie in der produkteigenen Dokumentation der Datenbank (im Falle der produkteigenen API der Datenbank) beschrieben.

System testen
Nur durch Testläufe können Sie feststellen, ob Ihre Anwendung und Ihr System korrekt konfiguriert sind. Die Systemkonfiguration (die erfolgreiche Kommunikation zwischen Warteschlangenmanager und Datenbank) kann durch Build und Ausführung eines der bereitgestellten Beispielprogramme getestet werden.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Db2 konfigurieren
DB2-Support- und -Konfigurationsinformationen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Die unterstützten Versionen von Db2 werden auf der Seite IBM WebSphere MQ -Detaillierte Systemvoraussetzungen definiert.

Anmerkung: 32-Bit-Instanzen von Db2 werden auf Plattformen mit einer 64-Bit-Version des Warteschlangenmanagers nicht unterstützt.

Gehen Sie wie folgt vor:

1. Prüfen Sie die Einstellungen der Umgebungsvariablen.
2. Erstellen Sie die Db2-Switchloaddatei.
3. Fügen Sie Konfigurationsinformationen für den Ressourcenmanager hinzu.
4. Ändern Sie bei Bedarf die Db2-Konfigurationsparameter.

Lesen Sie diese Informationen zusammen mit den allgemeinen Angaben im Abschnitt „System für die Datenbankkoordination konfigurieren” auf Seite 49.

Warnung: Wenn Sie `db2profile` auf den Plattformen UNIX and Linux ausführen, werden die Umgebungsvariablen LIBPATH und LD_LIBRARY_PATH definiert. Es wird empfohlen, `unset` diese Umgebungsvariablen zu verwenden. Weitere Informationen finden Sie im entsprechenden Handbuch *Einstieg* .

Einstellungen der Db2-Umgebungsvariablen überprüfen

Stellen Sie sicher, dass die Db2-Umgebungsvariablen für die Warteschlangenmanagerprozesse ***und*** in den Anwendungsprozessen gesetzt sind. Insbesondere muss die Umgebungsvariable DB2INSTANCE immer ***vor*** dem Start des Warteschlangenmanagers definiert werden. Die Umgebungsvariable DB2INSTANCE gibt die Db2-Instanz an, die die Db2-Datenbanken enthält, die aktualisiert werden. Beispiel:

- Verwenden Sie auf UNIX and Linux-Systemen folgende Angabe:

```
export DB2INSTANCE=db2inst1
```

- Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set DB2INSTANCE=DB2
```

Unter Windows mit einer Db2-Datenbank müssen Sie der Gruppe DB2USERS den Benutzer MUSR_MQADMIN hinzufügen, damit der Warteschlangenmanager gestartet werden kann.

Db2-Switchloaddatei erstellen

Am einfachsten kann die Db2-Switchloaddatei mithilfe der Beispieldatei 'xaswit.mak' erstellt werden, die von WebSphere MQ zur Erstellung der Switchloaddateien für verschiedene Datenbankprodukte bereitgestellt wird.

Auf Windows-Systemen ist die Datei 'xaswit.mak' im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm` zu finden.*MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist. Verwenden Sie den folgenden Befehl, um die Db2-Switchloaddatei mit Microsoft Visual C++ zu erstellen:

```
nmake /f xaswit.mak db2swit.dll
```

Die generierte Switchdatei wird in das Verzeichnis `c:\Program Files\IBM\WebSphere MQ\exits` gestellt.

Sie finden xaswit.mak im Verzeichnis *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.

Bearbeiten Sie die Datei 'xaswit.mak', indem Sie die Kommentarzeichen für die Zeilen *entfernen*, die der von Ihnen verwendeten Db2-Version entsprechen. Führen Sie anschließend die Makedatei mit dem folgenden Befehl aus:

```
make -f xaswit.mak db2swit
```

Die generierte 32-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits` gestellt.

Die generierte 64-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits64` gestellt.

Konfigurationsinformationen des Ressourcenmanagers für Db2 hinzufügen

Sie müssen die Konfigurationsinformationen für den Warteschlangenmanager ändern, damit Db2 als Beteiligter bei globalen Arbeitseinheiten deklariert werden kann. Die entsprechende Änderung der Konfigurationsinformationen wird im Abschnitt „Konfigurationsinformationen zum Warteschlangenmanager hinzufügen” auf Seite 51 ausführlicher beschrieben.

- Verwenden Sie auf Windows -und Linux -Systemen (x86 -und x86-64 -Plattformen) den WebSphere MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an.
- Geben Sie bei allen anderen Systemen die Details zu der Switchloaddatei in der XAResourceManager-Zeilengruppe in der Datei 'qm.ini' des Warteschlangenmanagers an.

Abbildung 9 auf Seite 55 ist ein UNIX-Beispiel mit einem XAResourceManager-Eintrag, bei dem die zu koordinierende Datenbank die Bezeichnung `mydbname` trägt, wobei dieser Name in `XAOpenString` angegeben wird:

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=mydbname,myuser,mypasswd,toc=t
  ThreadOfControl=THREAD
```

Abbildung 9. XAResourceManager-Beispieleintrag für Db2 auf UNIX-Plattformen

Anmerkung:

1. `ThreadOfControl=THREAD` kann nicht mit Db2 -Versionen vor Version 8 verwendet werden. Legen Sie `ThreadOfControl` und den XAOpenString-Parameter `toc` auf eine der folgenden Kombinationen fest:
   - `ThreadOfControl=THREAD` und `toc=t`
   - `ThreadOfControl=PROCESS` und `toc=p`

   Wenn Sie die XA-Switchloaddatei 'jdbcdb2' verwenden, um die JDBC/JTA-Koordination zu ermöglichen, müssen Sie `ThreadOfControl=PROCESS` und `toc=p` verwenden.

Db2-Konfigurationsparameter ändern

Sie müssen für jede Db2-Datenbank, die vom Warteschlangenmanager koordiniert wird, die Datenbankberechtigungen festlegen, den Parameter 'tp_mon_name' ändern und den Parameter 'maxappls' zurücksetzen. Gehen Sie hierfür wie folgt vor:

Legen Sie Datenbankberechtigungen fest

Die Warteschlangenmanagerprozesse werden auf Systemen mit UNIX and Linux mit dem ausführenden Benutzer und der Gruppe 'mqm' ausgeführt. Auf Windows-Systemen werden sie unter dem Benutzer ausgeführt, der den Warteschlangenmanager gestartet hat. Hierbei kann es sich um einen der folgenden Benutzer handeln:

1. Benutzer, der den Befehl `strmqm` ausgegeben hat, oder

2. Benutzer, unter dem der IBM MQSeries Service COM-Server ausgeführt wird

Standardmäßig heißt dieser Benutzer MUSR_MQADMIN.

Wenn Sie in der Zeichenfolge 'xa_open' keinen Benutzernamen und kein Kennwort angegeben haben, wird **der Benutzer, unter dem der Warteschlangenmanager ausgeführt wird**, von DB2 zur Authentifizierung des Aufrufs 'xa_open' verwendet. Verfügt dieser Benutzer (z. B. der Benutzer 'mqm' auf Systemen mit UNIX and Linux) nicht über minimale Berechtigungen in der Datenbank, verweigert die Datenbank die Authentifizierung des Aufrufs 'xa_open'.

Dieselben Überlegungen gelten auch für Ihren Anwendungsprozess. Wenn Sie in der Zeichenfolge 'xa_open' keinen Benutzernamen und kein Kennwort angegeben haben, authentifiziert Db2 den xa_open-Aufruf während des ersten MQBEGIN-Aufrufs mithilfe des Benutzers, unter dem Ihre Anwendung ausgeführt wird. Auch hier funktioniert dies nur, wenn der betreffende Benutzer über minimale Berechtigungen in der Datenbank verfügt.

Erteilen Sie beispielsweise dem Benutzer 'mqm' Verbindungsberechtigung in der Datenbank 'mydbname', indem Sie die folgenden Db2-Befehle ausgeben:

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

Weitere Informationen zur Sicherheit finden Sie im Abschnitt „Sicherheitsaspekte” auf Seite 64.

Windows **Ändern Sie den Parameter TP_MON_NAME.**

Nur bei Db2 für Windows-Systeme: Setzen Sie den Konfigurationsparameter TP_MON_NAME auf den Namen der DLL, mit der Db2 den Warteschlangenmanager für die dynamische Registrierung aufruft.

Geben Sie mit dem Befehl `db2 update dbm cfg using TP_MON_NAME mqmax` MQMAX.DLL als die Bibliothek an, mit der Db2 den Warteschlangenmanager aufruft. Diese muss sich in einem Verzeichnis innerhalb von PATH befinden.

Setzen Sie den Parameter 'maxappls' zurück

Unter Umständen müssen Sie die Einstellung für den Parameter *maxappls* prüfen, mit dem die maximale Anzahl an Anwendungen begrenzt wird, die mit einer Datenbank verbunden werden können. Weitere Informationen finden Sie im Thema „Installation und Konfiguration des Datenbankprodukts” auf Seite 50.

Oracle konfigurieren

Oracle-Support- und -Konfigurationsinformationen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Führen Sie die folgenden Schritte aus:

1. Prüfen Sie die Einstellungen der Umgebungsvariablen.
2. Erstellen Sie die Oracle-Switchloaddatei.
3. Fügen Sie Konfigurationsinformationen für den Ressourcenmanager hinzu.
4. Ändern Sie gegebenenfalls die Oracle-Konfigurationsparameter.

Eine aktuelle Liste der von IBM WebSphere MQ unterstützten Oracle-Versionen finden Sie auf der Seite IBM WebSphere MQ -Detaillierte Systemvoraussetzungen.

Einstellungen der Oracle-Umgebungsvariablen prüfen

Vergewissern Sie sich, dass Ihre Oracle-Umgebungsvariablen sowohl für Warteschlangenmanagerprozesse als auch in Ihren Anwendungsprozessen definiert sind. Legen Sie insbesondere immer die folgenden Umgebungsvariablen fest, bevor Sie den Warteschlangenmanager starten:

ORACLE_HOME
Das Oracle-Ausgangsverzeichnis. Verwenden Sie beispielsweise auf UNIX and Linux-Systemen folgende Angabe:

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```

Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set ORACLE_HOME=c:\oracle\ora81
```

ORACLE_SID
Die verwendete Oracle-SID. Wenn Sie für die Client/Server-Konnektivität Net8 verwenden, müssen Sie diese Umgebungsvariable möglicherweise nicht festlegen. Lesen Sie dies in Ihrer Oracle-Dokumentation nach.

Nachfolgend finden Sie ein Beispiel für die Definition dieser Umgebungsvariablen auf Systemen mit UNIX and Linux:

```
export ORACLE_SID=sid1
```

Die äquivalente Angabe auf Windows-Systemen lautet wie folgt:

```
set ORACLE_SID=sid1
```

Anmerkung: Die Umgebungsvariable PATH muss so festgelegt werden, dass sie das Verzeichnis mit Binärdateien (z. B. `ORACLE_INSTALL_DIR/VERSION/32BIT_NAME/bin` oder `ORACLE_INSTALL_DIR/VERSION/64BIT_NAME/bin`) enthält. Andernfalls wird möglicherweise die Nachricht angezeigt, dass in der Maschine oraclient-Bibliotheken fehlen.

Bei Ausführung von Warteschlangenmanagern auf Windows-64-Bit-Systemen müssen 64-Bit- und 32-Bit-Oracle-Clients installiert werden. Die Installation beider Clients ist erforderlich, da der Warteschlangenmanager in Form von 32-Bit-Prozessen unter Verwendung einer 32-Bit-Switchloaddatei ausgeführt wird, die wiederum eine 32-Bit-Oracle-Client-DLL starten müssen.

Die von 64-Bit-Warteschlangenmanagern geladene Switchloaddatei muss auf die Oracle-64-Bit-Clientbibliotheken zugreifen. 32-Bit-Warteschlangenmanager müssen auf den 32-Bit-Oracle-Client zugreifen, wenn IBM WebSphere MQ auf einem Windows-64-Bit-System ausgeführt wird.

Oracle-Switchloaddatei erstellen

Verwenden Sie zum Erstellen der Oracle-Switchloaddatei die Beispieldatei `xaswit.mak`, die von IBM WebSphere MQ zum Erstellen von Switchloaddateien für verschiedene Datenbankprodukte bereitgestellt wird. Auf Windows -Systemen finden Sie `xaswit.mak` im Verzeichnis `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. Verwenden Sie Folgendes, um die Oracle -Switchloaddatei mit Microsoft Visual C++ zu erstellen: `nmake /f xaswit.mak oraswit.dll`

Die generierte Switchdatei wird in *MQ_INSTALLATION_PATH*`\exits`gespeichert.*MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.

Sie finden `xaswit.mak` im Verzeichnis *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.

Bearbeiten Sie die Datei `xaswit.mak` und entfernen Sie die Kommentarzeichen für die entsprechenden Zeilen der von Ihnen verwendeten Oracle-Version. Führen Sie anschließend die Makedatei mit dem folgenden Befehl aus:

```
make -f xaswit.mak oraswit
```

Die generierte 32-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits` gestellt.

Die generierte 64-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits64` gestellt.

Ressourcenmanager-Konfigurationsinformationen für Oracle hinzufügen

Sie müssen die Konfigurationsinformationen für den Warteschlangenmanager ändern und die Beteiligung von Oracle an globalen Arbeitseinheiten deklarieren. Die entsprechende Änderung der Konfigurationsinformationen für den Warteschlangenmanager wird im Abschnitt „Konfigurationsinformationen zum Warteschlangenmanager hinzufügen” auf Seite 51 ausführlicher beschrieben.

- Verwenden Sie unter Windows und Linux (x86- und x86-64-Plattformen) IBM WebSphere MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an.
- Geben Sie bei allen anderen Systemen die Details zu der Switchloaddatei in der XAResourceManager-Zeilengruppe in der Datei `qm.ini` des Warteschlangenmanagers an.

Abbildung 10 auf Seite 58 ist ein Beispiel für UNIX and Linux-Systeme mit einem XAResourceManager-Eintrag. Der XA-Zeichenfolge zum Öffnen muss `LogDir` hinzugefügt werden, damit alle Fehler- und Traceinformationen an derselben Stelle protokolliert werden.

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

Abbildung 10. XAResourceManager-Beispieleintrag für Oracle auf UNIX and Linux-Plattformen

Anmerkung:

1. In Abbildung 10 auf Seite 58 wurde die Zeichenfolge 'xa_open' mit vier Parametern verwendet. Weitere Parameter können wie in der Oracle-Dokumentation beschrieben hinzugefügt werden.
2. Bei Verwendung des IBM WebSphere MQ-Parameters `ThreadOfControl=THREAD` müssen Sie den Oracle-Parameter `+threads=true` in der XAResourceManager-Zeilengruppe verwenden.

Weitere Informationen zur Zeichenfolge 'xa_open' finden Sie im *Oracle8 Server Application Developer's Guide*.

Oracle-Konfigurationsparameter ändern

Für jede vom Warteschlangenmanager koordinierte Oracle-Datenbank muss die maximale Sitzungsanzahl geprüft werden und es müssen Datenbankberechtigungen festgelegt werden. Gehen Sie hierfür wie folgt vor:

Prüfen Sie die maximale Sitzungsanzahl
Möglicherweise müssen Sie die Einstellungen LICENSE_MAX_SESSIONS und PROCESSES prüfen und dabei die von den zum Warteschlangenmanager gehörigen Prozessen benötigten zusätzlichen Verbindungen berücksichtigen. Weitere Informationen finden Sie in „Installation und Konfiguration des Datenbankprodukts” auf Seite 50.

Legen Sie Datenbankberechtigungen fest
Der in der Zeichenfolge 'xa_open' angegebene Oracle-Benutzername muss über Zugriffsberechtigung für die Ansicht DBA_PENDING_TRANSACTIONS verfügen, wie in der Oracle-Dokumentation beschrieben.

Die erforderliche Berechtigung kann mit dem folgenden Beispielbefehl erteilt werden:

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

Informix konfigurieren

Informix-Support- und Konfigurationsinformationen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Führen Sie die folgenden Schritte aus:

1. Stellen Sie sicher, dass Sie das entsprechende Informix Client SDK installiert haben:
   - Für 32-Bit-Warteschlangenmanager und -Anwendungen wird ein 32-Bit-Informix Client SDK benötigt.
   - Für 64-Bit-Warteschlangenmanager und -Anwendungen wird ein 64-Bit-Informix Client SDK benötigt.
2. Stellen Sie sicher, dass die Informix-Datenbanken ordnungsgemäß erstellt wurden.
3. Prüfen Sie die Einstellungen der Umgebungsvariablen.
4. Erstellen Sie die Informix-Switchloaddatei.
5. Fügen Sie Konfigurationsinformationen für den Ressourcenmanager hinzu.

Eine aktuelle Liste der Versionen von Informix , die von WebSphere MQ unterstützt werden, finden Sie auf der SeiteIBM WebSphere MQ -Detaillierte Systemvoraussetzungen .

Ordnungsgemäße Erstellung der Informix-Datenbanken sicherstellen

Jede Informix-Datenbank, die von einem WebSphere MQ-Warteschlangenmanager koordiniert werden soll, muss mit dem Parameter `log` erstellt werden. Beispiel:

```
create database mydbname with log;
```

Informix-Datenbanken, bei deren Erstellung der Parameter `log` nicht angegeben wurde, können von WebSphere MQ-Warteschlangenmanagern nicht koordiniert werden. Wenn ein Warteschlangenmanager versucht, eine Informix-Datenbank zu koordinieren, bei deren Erstellung der Parameter `log` nicht angegeben wurde, schlägt der Aufruf 'xa_open' für Informix fehl und es wird eine Reihe von FFST-Fehlern generiert.

Einstellungen der Informix-Umgebungsvariablen prüfen

Stellen Sie sicher, dass Ihre Informix-Umgebungsvariablen sowohl für Warteschlangenmanagerprozesse ***als auch in*** Ihren Anwendungsprozessen definiert sind. Legen Sie insbesondere immer die folgenden Umgebungsvariablen fest, **bevor** Sie den Warteschlangenmanager starten:

INFORMIXDIR
Das Verzeichnis der Informix-Produktinstallation.

- Verwenden Sie bei 32-Bit-Anwendungen unter UNIX and Linux den folgenden Befehl:

  ```
  export INFORMIXDIR=/opt/informix/32-bit
  ```

- Verwenden Sie bei 64-Bit-Anwendungen unter UNIX and Linux den folgenden Befehl:

  ```
  export INFORMIXDIR=/opt/informix/64-bit
  ```

- Verwenden Sie bei Windows-Anwendungen den folgenden Befehl:

  ```
  set INFORMIXDIR=c:\informix
  ```

Bei Systemen mit 64-Bit-Warteschlangenmanagern, die sowohl 32- als auch 64-Bit-Anwendungen unterstützen müssen, müssen sowohl das Informix Client SDK mit 32 Bit als auch das Informix Client SDK mit 64 Bit installiert sein. Mit der zum Erstellen einer Switchloaddatei verwendeten Beispielmakefile `xaswit.mak` werden ebenfalls beide Produktinstallationsverzeichnisse festgelegt.

INFORMIXSERVER
Der Name des Informix-Servers. Verwenden Sie zum Beispiel auf Systemen mit UNIX and Linux:

```
export INFORMIXSERVER=hostname_1
```

Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG
Der Name der Informix-Serverkonfigurationsdatei. Verwenden Sie zum Beispiel auf Systemen mit UNIX and Linux:

```
export ONCONFIG=onconfig.hostname_1
```

Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set ONCONFIG=onconfig.hostname_1
```

Informix-Switchloaddatei erstellen

Verwenden Sie zum Erstellen der Informix-Switchloaddatei die Beispieldatei 'xaswit.mak', die von WebSphere MQ zum Erstellen von Switchloaddateien für verschiedene Datenbankprodukte bereitgestellt wird. Auf Windows-Systemen ist die Datei 'xaswit.mak' im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm` zu finden.*MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist. Verwenden Sie zum Erstellen der Informix-Switchloaddatei mit Microsoft Visual C++ folgende Angabe:

```
nmake /f xaswit.mak infswit.dll
```

Die generierte Switchdatei wird in das Verzeichnis `c:\Program Files\IBM\WebSphere MQ\exits`gestellt.

Sie finden xaswit.mak im Verzeichnis *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.

Bearbeiten Sie die Datei 'xaswit.mak' und *entfernen Sie die Kommentarzeichen* für die entsprechenden Zeilen der von Ihnen verwendeten Informix-Version. Führen Sie anschließend die Makedatei mit dem folgenden Befehl aus:

```
make -f xaswit.mak infswit
```

Die generierte 32-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits` gestellt.

Die generierte 64-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits64` gestellt.

Ressourcenmanager-Konfigurationsinformationen für Informix hinzufügen

Sie müssen die Konfigurationsinformationen für den Warteschlangenmanager ändern, um die Beteiligung von Informix an globalen Arbeitseinheiten zu deklarieren. Die entsprechende Änderung der Konfigurationsinformationen für den Warteschlangenmanager wird im Abschnitt „Konfigurationsinformationen zum Warteschlangenmanager hinzufügen” auf Seite 51 ausführlicher beschrieben.

- Verwenden Sie auf Windows -und Linux -Systemen (x86 -und x86-64 -Plattformen) den WebSphere MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an.
- Geben Sie bei allen anderen Systemen die Details zu der Switchloaddatei in der XAResourceManager-Zeilengruppe in der Datei 'qm.ini' des Warteschlangenmanagers an.

Abbildung 11 auf Seite 61 ist ein UNIX-Beispiel mit einem XAResourceManager-Eintrag in der Datei 'qm.ini', bei dem die zu koordinierende Datenbank die Bezeichnung `mydbname` trägt, wobei dieser Namen in `XAOpenString` angegeben wird:

```
XAResourceManager:
  Name=myinformix
  SwitchFile=infswit
  XAOpenString=DB=mydbname@myinformixserver\;USER=myuser\;PASSWD=mypasswd
  ThreadOfControl=THREAD
```

Abbildung 11. XAResourceManager-Beispieleintrag für Informix auf UNIX-Plattformen

Anmerkung: Standardmäßig wird mit der Beispieldatei 'xaswit.mak' auf UNIX-Plattformen eine Switchloaddatei erstellt, die Informix-Threadbibliotheken verwendet. Bei Verwendung dieser Informix-Bibliotheken muss sichergestellt sein, dass 'ThreadOfControl' auf THREAD gesetzt ist. In Abbildung 11 auf Seite 61 ist in der XAResourceManager-Zeilengruppe in der Datei 'qm.ini' das Attribut 'ThreadOfControl' auf THREAD gesetzt. Bei Angabe von THREAD müssen Anwendungen unter Verwendung der Informix-Threadbibliotheken sowie der API-Threadbibliotheken von WebSphere MQ erstellt werden.

Im Attribut 'XAOpenString' muss der Datenbankname gefolgt vom Symbol @ und anschließend dem Informix-Servernamen angegeben sein.

Für die Verwendung der Informix-Bibliotheken ohne Threads muss sichergestellt sein, dass das Attribut 'ThreadOfControl' in der XAResourceManager-Zeilengruppe der Datei 'qm.ini' auf PROCESS gesetzt ist. Außerdem müssen in der Beispieldatei 'xaswit.mak' folgende Änderungen vorgenommen werden:

1. Die Kommentarzeichen für das Erstellen einer Switchloaddatei ohne Threads muss entfernt werden.
2. Das Erstellen der Switchloaddatei mit Threads muss auf Kommentar gesetzt werden.

Sybase-Konfiguration

Sybase-Support- und -Konfigurationsinformationen

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Führen Sie die folgenden Schritte aus:

1. Vergewissern Sie sich, dass Sie die Sybase-XA-Bibliotheken installiert haben, und installieren Sie dazu beispielsweise die XA-DTM-Option.
2. Prüfen Sie die Einstellungen der Umgebungsvariablen.
3. Aktivieren Sie die Sybase-XA-Unterstützung.
4. Erstellen Sie die Sybase-Switchloaddatei.
5. Fügen Sie Konfigurationsinformationen für den Ressourcenmanager hinzu.

Eine aktuelle Liste der von WebSphere MQ unterstützten Sybase-Ebenen finden Sie auf der Seite IBM WebSphere MQ -Detaillierte Systemvoraussetzungen.

Einstellungen der Sybase-Umgebungsvariablen prüfen

Vergewissern Sie sich, dass Ihre Sybase-Umgebungsvariablen sowohl für Warteschlangenmanagerprozesse ***als auch in*** Ihren Anwendungsprozessen definiert sind. Legen Sie insbesondere immer die folgenden Umgebungsvariablen fest, **bevor** Sie den Warteschlangenmanager starten:

SYBASE
Die Speicherposition der Sybase-Produktinstallation. Verwenden Sie zum Beispiel auf Systemen mit UNIX and Linux:

```
export SYBASE=/sybase
```

Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set SYBASE=c:\sybase
```

SYBASE_OCS
Das Verzeichnis unter SYBASE, in dem Sie die Sybase-Clientdateien installiert haben. Verwenden Sie zum Beispiel auf Systemen mit UNIX and Linux:

```
export SYBASE_OCS=OCS-12_0
```

Verwenden Sie auf Windows-Systemen folgende Angabe:

```
set SYBASE_OCS=OCS-12_0
```

Sybase-XA-Unterstützung aktivieren

Definieren Sie in der Sybase-XA-Konfigurationsdatei $SYBASE/$SYBASE_OCS/xa_config einen LRM (Logical Resource Manager) für jede Verbindung zu dem zu aktualisierenden Sybase-Server. Ein Beispiel für den Inhalt der Datei $SYBASE/$SYBASE_OCS/xa_config finden Sie in Abbildung 12 auf Seite 62.

```
# The first line must always be a comment
[xa]
  LRM=lrmname
  server=servername
```

Abbildung 12. Beispielinhalt von $SYBASE/$SYBASE_OCS/xa_config

Sybase-Switchloaddatei erstellen

Verwenden Sie zum Erstellen der Sybase-Switchloaddatei die mit WebSphere MQ bereitgestellten Beispieldateien.Auf Windows -Systemen finden Sie xaswit.mak im Verzeichnis C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm. Verwenden Sie zum Erstellen der Sybase-Switchloaddatei mit Microsoft Visual C++ folgende Angabe:

```
nmake /f xaswit.mak sybswit.dll
```

Die generierte Switchdatei wird in das Verzeichnis c:\Program Files\IBM\WebSphere MQ\exitsgestellt.

Sie finden xaswit.mak im Verzeichnis *MQ_INSTALLATION_PATH*/samp/xatm. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.

Bearbeiten Sie die Datei 'xaswit.mak' und *entfernen Sie die Kommentarzeichen* für die entsprechenden Zeilen der von Ihnen verwendeten Version. Führen Sie anschließend die Makedatei mit dem folgenden Befehl aus:

```
make -f xaswit.mak sybswit
```

Die generierte 32-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits` gestellt.

Die generierte 64-Bit-Switchloaddatei wird in das Verzeichnis `/var/mqm/exits64` gestellt.

Ressourcenmanager-Konfigurationsinformationen für Sybase hinzufügen

Sie müssen die Konfigurationsinformationen für den Warteschlangenmanager ändern und die Beteiligung von Sybase an globalen Arbeitseinheiten deklarieren. Die Änderung der Konfigurationsinformationen wird im Abschnitt „Konfigurationsinformationen zum Warteschlangenmanager hinzufügen” auf Seite 51 ausführlicher beschrieben.

- Verwenden Sie auf Windows -und Linux -Systemen (x86 -und x86-64 -Plattformen) den WebSphere MQ Explorer. Geben Sie die Details zu der Switchloaddatei in der Eigenschaftenanzeige für den Warteschlangenmanager unter dem XA-Ressourcenmanager an.
- Geben Sie bei allen anderen Systemen die Details zu der Switchloaddatei in der XAResourceManager-Zeilengruppe in der Datei 'qm.ini' des Warteschlangenmanagers an.

In Abbildung 13 auf Seite 63 ist ein Beispiel unter UNIX and Linux dargestellt, in dem die der LRM-Definition *lrmname* in der Sybase-XA-Konfigurationsdatei $SYBASE/$SYBASE_OCS/`xa_config` zugeordnete Datenbank verwendet wird. Geben Sie auch einen Protokolldateinamen an, wenn die XA-Funktionsaufrufe protokolliert werden sollen:

```
XAResourceManager:
  Name=mysybase
  SwitchFile=sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
  ThreadOfControl=THREAD
```

Abbildung 13. XAResourceManager-Beispieleintrag für Sybase auf den Plattformen UNIX and Linux

Multithread-Programme mit Sybase verwenden

Wenn Sie Multithread-Programme mit globalen WebSphere MQ-Arbeitseinheiten verwenden, die Sybase-Aktualisierungen beinhalten, ***müssen*** Sie für den Parameter 'ThreadOfControl' den Wert THREAD verwenden. Achten Sie außerdem darauf, dass Sie Ihr Programm (und die Switchloaddatei) mit den threadsicheren Sybase-Bibliotheken (_r-Versionen) verknüpfen. Die Verwendung des Wertes THREAD für den Parameter 'ThreadOfControl' ist in Abbildung 13 auf Seite 63 dargestellt.

Mehrere Datenbankkonfigurationen

Wenn Sie den Warteschlangenmanager so konfigurieren möchten, dass Aktualisierungen mehrerer Datenbanken in globale Arbeitseinheiten einbezogen werden können, fügen Sie für jede Datenbank eine XAResourceManager-Zeilengruppe hinzu.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Falls die Datenbanken alle von demselben Datenbankmanager verwaltet werden, wird in jeder Zeilengruppe eine andere Datenbank definiert. Zwar ist in jeder Zeilengruppe dieselbe *Switchdatei* angegeben, die Inhalte von *XAOpenString* sind jedoch unterschiedlich, da hier der Name der zu aktualisierenden Datenbank angegeben ist. Mit den in Abbildung 14 auf Seite 64 dargestellten Zeilengruppen beispielsweise wird der Warteschlangenmanager mit den DB2-Datenbanken *MQBankDB* und *MQFeeDB* auf UNIX and Linux-Systemen konfiguriert.

Wichtig: Es können nicht mehrere Zeilengruppen vorhanden sein, die auf die gleiche Datenbank verweisen. Diese Konfiguration kann nicht funktionieren und wird bei einem Konfigurationsversuch fehlschlagen.

Sie erhalten Fehler im Format `when the MQ code makes its second xa_open call in any process in this environment, the database software fails the second xa_open with a -5 error, XAER_INVAL`.

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=DB2 MQFeeDB
  SwitchFile=db2swit
  XAOpenString=MQFeeDB
```

Abbildung 14. XAResourceManager-Beispieleinträge für mehrere Db2-Datenbanken

Falls die zu aktualisierenden Datenbanken von verschiedenen Datenbankmanagern aktualisiert werden, fügen Sie jeweils eine XAResourceManager-Zeilengruppe hinzu. In diesem Fall ist in jeder Zeilengruppe eine andere *Switchdatei* angegeben. Wird *MQFeeDB* beispielsweise von Oracle und nicht von DB2 verwaltet, verwenden Sie auf UNIX and Linux-Systemen die folgenden Zeilengruppen:

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=Oracle MQFeeDB
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypassword+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

Abbildung 15. XAResourceManager-Beispieleinträge für eine DB2-Datenbank und eine Oracle-Datenbank

Im Prinzip können beliebig viele Datenbankinstanzen mit einem einzelnen Warteschlangenmanager konfiguriert werden.

Anmerkung: Informationen zur Unterstützung für die Einbeziehung von Informix-Datenbanken in mehrere Datenbankaktualisierungen innerhalb globaler Arbeitseinheiten finden Sie in der Readme-Datei zu dem Produkt.

Sicherheitsaspekte

Aspekte der Datenbankausführung unter dem XA-Modell.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Die folgenden Informationen sollen nur zur Orientierung dienen. Stellen Sie in allen Fällen anhand der mit dem Datenbankmanager bereitgestellten Dokumentation fest, wie sich die Ausführung Ihrer Datenbank unter dem XA-Modell auf die Sicherheit auswirkt.

In Anwendungsprozessen wird der Beginn einer globalen Arbeitseinheit mit dem Verb `MQBEGIN` bezeichnet. Beim ersten von einer Anwendung ausgegebenen Aufruf `MQBEGIN` wird durch Aufruf des entsprechenden Clientbibiliotekcodes am xa_open-Einstiegspunkt eine Verbindung zu allen beteiligten Datenbanken hergestellt. Alle Datenbankmanager bieten einen Mechanismus zur Angabe einer Benutzer-ID und eines Kennworts in ihrer XA-Zeichenfolge zum Öffnen (XAOpenString). Dies ist der einzige Zeitpunkt, an dem Authentifizierungsdaten fließen.

Beachten Sie, dass Fastpath-Anwendungen auf den Plattformen UNIX and Linux mit der effektiven Benutzer-ID 'mqm' ausgeführt werden müssen, wenn MQI-Aufrufe erfolgen.

Überlegungen zur Vorgehensweise bei Verlust des Kontakts zum XA-Ressourcenmanager

Der Warteschlangenmanager akzeptiert, wenn Datenbankmanager nicht verfügbar sind. Das heißt, Sie können den Warteschlangenmanager unabhängig vom Datenbankserver starten und stoppen. Sobald der Kontakt wiederhergestellt ist, werden Warteschlangenmanager und Datenbank resynchronisiert. Sie können unbestätigte Arbeitseinheiten auch manuell mit dem Befehl rsvmqtrn auflösen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Für normale Operationen ist nach der Konfiguration nur wenig Verwaltung notwendig. Der Verwaltungsjob ist einfacher, da es der Warteschlangenmanager akzeptiert, wenn Datenbankmanager nicht verfügbar sind. Dies bedeutet insbesondere Folgendes:

- Der Warteschlangenmanager kann jederzeit gestartet werden, ohne zuvor die einzelnen Datenbankmanager starten zu müssen.
- Wenn einer der Datenbankmanager nicht mehr verfügbar ist, muss der Warteschlangenmanager nicht gestoppt und erneut gestartet werden.

Somit kann der Warteschlangenmanager unabhängig vom Datenbankserver gestartet und gestoppt werden.

Wenn der Kontakt zwischen dem Warteschlangenmanager und einer Datenbank verloren geht, muss eine Resynchronisation erfolgen, wenn beide wieder verfügbar sind. Resynchronisation bezeichnet den Prozess, mit dem unbestätigte Arbeitseinheiten, die diese Datenbank einbeziehen, abgeschlossen werden Im Allgemeinen geschieht dies automatisch ohne Benutzereingriff. Der Warteschlangenmanager fragt die Datenbank nach einer Liste der unbestätigten Arbeitseinheiten. Anschließend weist er die Datenbank an, die einzelnen unbestätigten Arbeitseinheiten festzuschreiben oder rückgängig zu machen.

Beim Start wird der Warteschlangenmanager mit den einzelnen Datenbanken resynchronisiert. Wenn eine einzelne Datenbank nicht mehr verfügbar ist, muss nur diese resynchronisiert werden, wenn der Warteschlangenmanager merkt, dass sie wieder verfügbar ist.

Der Kontakt zwischen dem Warteschlangenmanager und einer zuvor nicht verfügbaren Datenbank wird automatisch wiederhergestellt, wenn mit MQBEGIN neue globale Arbeitseinheiten gestartet werden. Hierfür wird die Funktion 'xa_open' in der Datenbankclientbibliothek aufgerufen. Falls dieser Aufruf 'xa_open' fehlschlägt, gibt MQBEGIN den Beendigungscode MQCC_WARNING und den Ursachencode MQRC_PARTICIPANT_NOT_AVAILABLE zurück. Sie können den Aufruf MQBEGIN zu einem späteren Zeitpunkt wiederholen.

Versuchen Sie den Vorgang nicht mehrmals mit einer globalen Arbeitseinheit, die Aktualisierungen einer Datenbank beinhaltet, welche während des Aufrufs MQBEGIN einen Fehler angezeigt hat. Es wird keine Verbindung zu dieser Datenbank geben, über die Aktualisierungen erfolgen können. Die einzige Möglichkeit besteht darin, das Programm zu beenden oder den Aufruf MQBEGIN von Zeit zu Zeit zu wiederholen, in der Hoffnung, dass die Datenbank möglicherweise wieder verfügbar wird.

Alternativ dazu können Sie auch mit dem Befehl `rsvmqtrn` alle unbestätigten Arbeitseinheiten explizit auflösen.

Unbestätigte Arbeitseinheiten

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Möglicherweise gibt es in einer Datenbank unbestätigte Arbeitseinheiten, wenn der Kontakt zum Warteschlangenmanager unterbrochen wird, nachdem der Datenbankmanager aufgefordert wurde, Vorbereitungen zu treffen. Bis der Datenbankserver vom Warteschlangenmanager das Ergebnis (Festschreibung oder Rollback) erhält, muss er die Datenbanksperren im Zusammenhang mit den Aktualisierungen beibehalten.

Da aufgrund dieser Sperren andere Anwendungen Datenbanksätze weder aktualisieren noch lesen können, muss so bald wie möglich eine Resynchronisation erfolgen.

Falls Sie die automatische Resynchronisation des Warteschlangenmanagers mit der Datenbank aus irgendeinem Grund nicht abwarten können, können Sie die Datanbankaktualisierungen mithilfe von Datenbankmanagerfunktionen manuell festschreiben bzw. rückgängig machen. In *X/Open Distributed Transaction Processing: The XA Specification* wird in diesem Zusammenhang auch von einer *heuristischen* Entscheidung gesprochen. Nutzen Sie diese Möglichkeit nur als letzten Abhilfemaßnahme, da dabei die Datenintegrität beeinträchtigt werden kann; so könnten Sie beispielsweise versehentlich die Datenbankaktualisierungen rückgängig machen, wenn alle anderen Beteiligten ihre Aktualisierungen festgeschrieben haben.

Weitaus besser ist es, den Warteschlangenmanager erneut zu starten oder nach dem Neustart der Datenbank mit dem Befehl `rsvmqtrn` eine automatische Resynchronisation einzuleiten.

Ausstehende Arbeitseinheiten mit dem Befehl 'dspmqtrn' anzeigen

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wenn ein Datenbankmanager nicht verfügbar ist, können Sie mit dem Befehl **dspmqtrn** den Status ausstehender globaler Arbeitseinheiten überprüfen, die diese Datenbank mit einbeziehen.

Mit dem Befehl **dspmqtrn** werden nur Arbeitseinheiten mit mindestens einem unbestätigten Beteiligten angezeigt. Die Beteiligten warten auf die Entscheidung des Warteschlangenmanagers über die Festschreibung der das Rollback der vorbereiteten Aktualisierungen.

Für jede dieser globalen Arbeitseinheiten wird in der Ausgabe von **dspmqtrn** der Status der einzelnen Beteiligten angezeigt. Falls die Arbeitseinheit die Ressourcen eines bestimmten Ressourcenmanagers nicht aktualisiert hat, wird sie nicht angezeigt.

Bezüglich einer unbestätigten Arbeitseinheit ist angegeben, dass ein Ressourcenmanager eine der folgenden Aktionen durchgeführt hat:

Vorbereitet
Der Ressourcenmanager ist bereit, die Aktualisierungen festzuschreiben.

Festgeschrieben
Der Ressourcenmanager hat die Aktualisierungen festgeschrieben.

Rückgängig gemacht
Der Ressourcenmanager hat die Aktualisierungen rückgängig gemacht.

Teilgenommen
Der Ressourcenmanager ist Beteiligter, hat jedoch die Aktualisierungen weder vorbereitet, noch festgeschrieben oder rückgängig gemacht.

Beim Neustart fragt der Warteschlangenmanager jede Datenbank mit einer XAResourceManager-Zeilengruppe nach einer Liste ihrer unbestätigten globalen Arbeitseinheiten. Falls die Datenbank nicht erneut gestartet wurde oder aus anderen Gründen nicht verfügbar ist, kann der Warteschlangenmanager die endgültigen Ergebnisse für die betreffenden Arbeitseinheiten noch nicht an die Datenbank übergeben. Das Ergebnis der unbestätigten Arbeitseinheiten wird der Datenbank bei der nächsten Gelegenheit übergeben, wenn die Datenbank wieder verfügbar ist.

In diesem Fall wird für den Datenbankmanager der Status *Vorbereitet* gemeldet, bis die Resynchronisation durchgeführt wurde.

Wenn vom Befehl `dspmqtrn` eine unbestätigte Arbeitseinheit angezeigt wird, werden zunächst alle möglicherweise beteiligten Ressourcenmanager aufgeführt. Diesen wird eine eindeutige Kennung, *RMId*, zugeordnet, die anstelle des *Namens* der Ressourcenmanager verwendet wird, wenn ihr Status in Bezug auf eine unbestätigte Arbeitseinheit gemeldet wird.

Im Abschnitt Beispielausgabe für 'dspmqtrn' ist das Ergebnis der folgenden Befehlsausgabe dargestellt:

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A50600002013746573 74]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Dabei ist *Transaction number* die ID der Transaktion, die mit dem Befehl rsvmqtrn verwendet werden kann. Weitere Informationen zur Nachricht AMQ7056 finden Sie im Thema AMQ7000-7999: WebSphere MQ-Produkt. Die *XID* -Variablen sind Teil der *X/Open XA-Spezifikation*. Aktuelle Informationen zu dieser Spezifikation finden Sie unter https://publications.opengroup.org/c193.

Abbildung 16. Beispielausgabe für 'dspmqtrn'

Aus der Ausgabe im Abschnitt Beispielausgabe für 'dspmqtrn' geht hervor, dass dem Warteschlangenmanager drei Ressourcenmanager zugeordnet sind. Bei dem ersten Ressourcenmanager 0 handelt es sich um den Warteschlangenmanager selbst. Die anderen beiden Ressourcenmanagerinstanzen sind die Datenbanken MQBankDB und MQFeeDB Db2 .

In dem Beispiel ist eine einzelne unbestätigte Arbeitseinheit dargestellt. Für alle drei Ressourcenmanager wird eine Nachricht ausgegeben, was bedeutet, dass Aktualisierungen am Warteschlangenmanager und an beiden Db2 -Datenbanken innerhalb der Arbeitseinheit vorgenommen wurden.

Die Aktualisierungen am Warteschlangenmanager (Ressourcenmanager **0**) wurden *festgeschrieben*. Die Aktualisierungen an den Db2 -Datenbanken befinden sich im Status *prepared* . Dies bedeutet, dass Db2 nicht mehr verfügbar sein muss, bevor es aufgerufen wurde, um die Aktualisierungen an den Datenbanken *MQBankDB* und *MQFeeDB* festzuschreiben.

Die unbestätigte Arbeitseinheit verfügt über eine externe Kennung, eine sogenannte XID (*Transaktions-ID*). Dies ist ein Teil der Daten, die Db2 vom Warteschlangenmanager übergeben werden, um seinen Teil der globalen Arbeitseinheit zu identifizieren.

Ausstehende Arbeitseinheiten mit dem Befehl 'rsvmqtrn' auflösen
Ausstehende Arbeitseinheiten werden bei der Resynchronisation von DB2 und dem Warteschlangenmanager abgeschlossen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

In der Ausgabe in Abbildung 16 auf Seite 67 ist eine einzelne unbestätigte Arbeitseinheit angegeben, bei der die Festschreibungsentscheidung noch an beide DB2-Datenbanken übermittelt werden muss.

Damit diese Arbeitseinheit abgeschlossen wird, müssen der Warteschlangenmanager und DB2 resynchronisiert werden, sobald DB2 wieder verfügbar wird. Der Warteschlangenmanager nutzt den Start neuer Arbeitseinheiten als Gelegenheit, den Kontakt zu DB2 wiederherzustellen. Alternativ dazu können Sie den Warteschlangenmanager auch zu einer expliziten Resynchronisation mit dem Befehl **rsvmqtrn** anweisen.

Führen Sie diese Maßnahme bald nach dem Neustart von DB2 durch, damit alle Datenbanksperren im Zusammenhang mit der unbestätigten Arbeitseinheit so schnell wie möglich freigegeben werden. Verwenden Sie die Option '-a', um den Warteschlangenmanager aufzufordern, alle unbestätigten Arbeits-

einheiten aufzulösen. Im folgenden Beispiel wurde DB2 erneut gestartet, der Warteschlangenmanager kann also die unbestätigte Arbeitseinheit auflösen:

```
> rsvmqtrn -m MY_QMGR -a
Any in-doubt transactions have been resolved.
```

Heterogene Ergebnisse und Fehler
Auch wenn der Warteschlangenmanager ein Protokoll für zweiphasige Festschreibung verwendet, lässt sich nicht völlig ausschließen, dass Arbeitseinheiten mit heterogenen Ergebnissen abgeschlossen werden. Hierzu kommt es, wenn einige Beteiligte ihre Aktualisierungen festschreiben, während andere sie zurücksetzen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wenn Arbeitseinheiten mit heterogenen Ergebnissen abgeschlossen werden, hat dies schwerwiegende Konsequenzen, da sich gemeinsam genutzte Ressourcen, die als einzelne Arbeitseinheit hätten aktualisiert werden müssen, nicht mehr in einem konsistenten Zustand befinden.

Zu heterogenen Ergebnissen kommt es vor allem, wenn heuristische Entscheidungen zu Arbeitseinheiten getroffen werden und der Warteschlangenmanager unbestätigte Arbeitseinheiten nicht selbst auflösen kann. Über diese Entscheidungen hat der Warteschlangenmanager keine Kontrolle.

Wenn der Warteschlangenmanager heterogene Ergebnisse feststellt, generiert er FFST-Informationen und dokumentiert den Fehler mit einer der beiden folgenden Nachrichten in seinen Fehlerprotokollen:

- Falls ein Datenbankmanager anstelle einer Festschreibung ein Rollback vornimmt:

```
AMQ7606 A transaction has been committed but one or more resource
        managers have rolled back.
```

- Falls ein Datenbankmanager anstelle eines Rollbacks eine Festschreibung durchführt:

```
AMQ7607 A transaction has been rolled back but one or more resource
        managers have committed.
```

In weiteren Nachrichten ist angegeben, welche Datenbanken aufgrund der heuristischen Entscheidung beschädigt sind. Es ist nun Ihre Aufgabe, die Konsistenz der betreffenden Datenbanken lokal wiederherzustellen. Hierbei handelt es sich um ein kompliziertes Verfahren, bei dem die fälschlicherweise festgeschriebene bzw. zurückgesetzte Aktualisierung zunächst isoliert und die Datenbankänderung anschließend manuell rückgängig gemacht bzw. wiederhergestellt werden muss.

Konfigurationsinformationen ändern
Ändern Sie die Ressourcenmanager-Konfigurationsinformationen nicht mehr, nachdem der Warteschlangenmanager erfolgreich mit der Koordination globaler Arbeitseinheiten begonnen hat.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wenn Sie die Konfigurationsinformationen ändern müssen, können Sie dies jederzeit tun, die Änderungen treten allerdings erst nach dem Neustart des Warteschlangenmanagers in Kraft.

Wenn Sie die Ressourcenmanager-Konfigurationsinformationen für eine Datenbank entfernen, nehmen Sie dem Warteschlangenmanager damit praktisch die Möglichkeit, Kontakt zu dem betreffenden Datenbankmanager herzustellen.

Ändern Sie ***nie*** das Attribut *Name* in Ihren Ressourcenmanager-Konfigurationsinformationen. Mit diesem Attribut wird die betreffende Datenbankmanagerinstanz dem Warteschlangenmanager gegenüber eindeutig angegeben. Wenn Sie diese eindeutige Kennung ändern, geht der Warteschlangenmanager davon aus, dass die Datenbank entfernt und eine komplett neue Instanz hinzugefügt wurde. Der Warteschlan-

genmanager ordnet ausstehende Arbeitseinheiten weiterhin dem alten *Namen* zu, sodass die Datenbank möglicherweise in einem unbestätigten Zustand bleibt.

Datenbankmanagerinstanzen entfernen
Wenn eine Datenbank permanent aus Ihrer Konfiguration entfernt werden muss, vergewissern Sie sich, dass die Datenbank nicht unbestätigt ist, bevor Sie den Warteschlangenmanager erneut starten.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Von den Datenbankprodukten werden Befehle zur Auflistung unbestätigter Transaktionen zur Verfügung gestellt. Ermöglichen Sie im Falle unbestätigter Transaktionen dem Warteschlangenmanager zunächst eine Resynchronisation mit der Datenbank. Starten Sie hierfür den Warteschlangenmanager. Mit dem Befehl **rsvmqtrn** oder dem datenbankeigenen Befehl zur Anzeige unbestätigter Arbeitseinheiten können Sie prüfen, ob die Resynchronisation tatsächlich erfolgt ist. Wenn Sie sicher sind, dass die Resynchronisation stattgefunden hat, beenden Sie den Warteschlangenmanager und entfernen Sie die Konfigurationsinformationen der Datenbank.

Wenn Sie sich nicht an diese Vorgehensweise halten, merkt sich der Warteschlangenmanager alle unbestätigten Arbeitseinheiten, die die betreffende Datenbank einbeziehen. Es wird dann jedes Mal, wenn der Warteschlangenmanager erneut gestartet wird, eine Warnung, AMQ7623, ausgegeben. Wenn Sie diese Datenbank überhaupt nicht mehr mit dem Warteschlangenmanager konfigurieren möchten, verwenden Sie im Befehl **rsvmqtrn** die Option '-r' und weisen Sie den Warteschlangenmanager damit an, die Beteiligung der Datenbank an den unbestätigten Transaktionen zu übergehen. Der Warteschlangenmanager übergeht solche Transaktionen nur, wenn die unbestätigten Transaktionen mit allen Beteiligten abgeschlossen wurden.

Es kann vorkommen, dass Ressourcenmanager-Konfigurationsinformationen vorübergehend entfernt werden müssen. Auf Systemen mit UNIX and Linux wird hierfür am besten die betreffende Zeilengruppe auf Kommentar gesetzt, sodass sie später leicht wiederhergestellt werden kann. Vielleicht entscheiden Sie sich für diese Vorgehensweise, wenn Sie immer Fehler erhalten, wenn der Warteschlangenmanager einen Kontakt zu einer bestimmten Datenbank bzw. einem Datenbankmanager herstellt. Wenn die betreffenden Ressourcenmanager-Konfigurationsinformationen vorübergehend entfernt werden, kann der Warteschlangenmanager die globalen Arbeitseinheiten für alle anderen Beteiligten starten. Nachfolgend sehen Sie ein Beispiel für eine auskommentierte XA`ResourceManager`-Zeilengruppe:

```
# This database has been temporarily removed
#XAResourceManager:
#  Name=mydb2
#  SwitchFile=db2swit
#  XAOpenString=mydbname,myuser,mypassword,toc=t
#  ThreadOfControl=THREAD
```

Abbildung 17. Auskommentierte XAResourceManager-Zeilengruppe auf Systemen mit UNIX and Linux

Verwenden Sie auf Windows-Systemen zum Löschen der Informationen über die Datenbankmanagerinstanz WebSphere MQ Explorer. Achten Sie ganz besonders darauf, bei der Wiederherstellung den richtigen Namen in das Feld *Name* einzugeben. Falls Sie den Namen falsch eingeben, kann es wie im Abschnitt „Konfigurationsinformationen ändern” auf Seite 68 beschrieben zu Fehlern bezüglich eines unbestätigten Status kommen.

Dynamische XA-Registrierung
Die XA-Spezifikation bietet eine Möglichkeit, die Anzahl der Aufrufe `xa_*` eines Transaktionsmanagers an einen Ressourcenmanager zu verringern. Diese Optimierung wird als *dynamische Registrierung* bezeichnet.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

DB2 unterstützt die dynamische Registrierung. Auch andere Datenbanken bieten möglicherweise Unterstützung hierfür; weitere Informationen finden Sie in der Dokumentation zu Ihrem Datenbankprodukt.

Warum ist die Optimierung durch dynamische Registrierung sinnvoll? Es kann sein, dass in Ihrer Anwendung einige globale Arbeitseinheiten Aktualisierungen von Datenbanktabellen umfassen, andere nicht. Wenn keine permanenten Aktualisierungen der Tabellen einer Datenbank vorgenommen wurden, muss die betreffende Datenbank nicht in das Festschreibungsprotokoll während MQCMIT aufgenommen werden.

Unabhängig davon, ob Ihre Datenbank die dynamische Registrierung unterstützt, ruft Ihre Anwendung `xa_open` während des ersten MQBEGIN-Aufrufs in einer WebSphere MQ -Verbindung auf. Im nachfolgenden Aufruf MQDISC ruft die Anwendung `xa_close` auf. Das Muster weiterer XA-Aufrufe hängt davon ab, ob die Datenbank die dynamische Registrierung unterstützt:

Falls Ihre Datenbank die dynamische Registrierung nicht unterstützt...
Jede globale Arbeitseinheit umfasst mehrere XA-Funktionsaufrufe, die von WebSphere MQ-Code an die Datenbankclientbibliothek erfolgen, wobei es keine Rolle spielt, ob Sie die Tabellen der betreffenden Datenbank innerhalb der Arbeitseinheit permanent aktualisiert haben. Hierzu gehören folgende Aufrufe:

- `xa_start` und `xa_end` aus dem Anwendungsprozess. Mit diesen Aufrufen werden Anfang und Ende einer globalen Arbeitseinheit deklariert.
- `xa_prepare`, `xa_commit` und `xa_rollback` aus dem Warteschlangenmanageragentenprozess 'amqzlaa0'. Mit diesen Aufrufen wird das Ergebnis der globalen Arbeitseinheit, also die Entscheidung über Festschreibung oder Rollback übermittelt.

Darüber hinaus ruft der Warteschlangenmanageragentenprozess beim ersten Aufruf MQBEGIN auch `xa_open` auf.

Falls Ihre Datenbank die dynamische Registrierung unterstützt...
Vom WebSphere MQ-Code werden nur die erforderlichen XA-Funktionsaufrufe durchgeführt. Bei einer globalen Arbeitseinheit, die **keine** permanenten Aktualisierungen von Datenbankressourcen einbezieht, erfolgen **keine** XA-Aufrufe an die Datenbank. Bei einer globalen Arbeitseinheit, die solche permanenten Aktualisierungen **einbezieht**, werden folgende Aufrufe durchgeführt:

- `xa_end` aus dem Anwendungsprozess, um das Ende der globalen Arbeitseinheit zu deklarieren.
- `xa_prepare`, `xa_commit` und `xa_rollback` aus dem Warteschlangenmanageragentenprozess 'amqzlaa0'. Mit diesen Aufrufen wird das Ergebnis der globalen Arbeitseinheit, also die Entscheidung über Festschreibung oder Rollback übermittelt.

Damit die dynamische Registrierung richtig funktioniert, muss die Datenbank die Möglichkeit haben, WebSphere MQ nach einer permanenten Aktualisierung mitzuteilen, dass sie in die aktuelle globale Arbeitseinheit einbezogen werden möchte. WebSphere MQ stellt zu diesem Zweck die Funktion `ax_reg` bereit.

Der in Ihrem Anwendungsprozess ausgeführte Client-Code der Datenbank findet die Funktion `ax_reg` und ruft sie auf, um die Tatsache, dass innerhalb der globalen Arbeitseinheit eine permanente Aktion erfolgt ist, *dynamisch zu registrieren*. Als Antwort auf diesen Aufruf `ax_reg` zeichnet WebSphere MQ die Beteiligung der Datenbank auf. Falls es sich um den ersten Aufruf `ax_reg` der betreffenden WebSphere MQ-Verbindung handelt, ruft der Warteschlangenmanageragentenprozess `xa_open` auf.

Dieser Aufruf `ax_reg` wird vom Code für den Datenbankclient durchgeführt, wenn der Code in Ihrem Prozess ausgeführt wird, also beispielsweise während eines Aufrufs SQL UPDATE oder eines anderen entsprechenden Aufrufs in der Client-API der Datenbank.

Fehlerbedingungen
Bei der dynamischen XA-Registrierung kann es zu einem irreführenden Fehler im Warteschlangenmanager kommen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Ein allgemeines Beispiel hierfür: wenn Sie vergessen, die Datenbankumgebungsvariablen vor dem Start des Warteschlangenmanagers ordnungsgemäß zu definieren, schlagen die Aufrufe `xa_open` des Warteschlangenmanagers fehl. Es können keine globalen Arbeitseinheiten verwendet werden.

Um dies zu vermeiden, sollten Sie sich vergewissern, dass die relevanten Umgebungsvariablen definiert sind, bevor Sie den Warteschlangenmanager starten. Beachten Sie die Informationen in der Dokumentation zu Ihrem Datenbankprodukt sowie die Ratschläge in den Abschnitten „Db2 konfigurieren" auf Seite 53, „Oracle konfigurieren" auf Seite 56 und „Sybase-Konfiguration" auf Seite 61.

Bei allen Datenbankprodukten ruft der Warteschlangenmanager `xa_open` einmal beim Warteschlangenmanagerstart im Rahmen der Wiederherstellungssitzung auf (wie im Abschnitt „Überlegungen zur Vorgehensweise bei Verlust des Kontakts zum XA-Ressourcenmanager" auf Seite 65 erläutert). Dieser Aufruf `xa_open` schlägt fehl, wenn Sie die Datenbankumgebungsvariablen nicht ordnungsgemäß definieren, der Warteschlangenmanager kann jedoch trotzdem gestartet werden. Dies liegt daran, dass mit demselben `xa_open`-Fehlercode von der Datenbankclientbibliothek angegeben wird, dass der Datenbankserver nicht verfügbar ist. In WebSphere MQ wird dies nicht als schwerwiegender Fehler behandelt, da es möglich sein muss, den Warteschlangenmanager zu starten, um die Verarbeitung von Daten außerhalb der globalen Arbeitseinheiten, die diese Datenbank einbeziehen, fortzusetzen.

Nachfolgende Aufrufe von `xa_open` werden vom Warteschlangenmanager während des ersten MQBEGIN an einer WebSphere MQ-Verbindung (wenn die dynamische Registrierung nicht genutzt wird) oder während eines Aufrufs der von WebSphere MQ bereitgestellten Funktion `ax_reg` durch den Datenbankclientcode (bei Nutzung der dynamischen Registrierung) durchgeführt.

Das **Timing** der Fehlerbedingungen (bzw. gelegentlich der FFST-Berichte) hängt davon ab, ob dynamische Registrierung genutzt wird:

- Bei Verwendung der dynamischen Registrierung könnte es sein, dass Ihr Aufruf MQBEGIN erfolgreich ist, der Datenbankaufruf SQL UPDATE (oder ähnlich) schlägt jedoch fehl.
- Wenn keine dynamische Registrierung verwendet wird, schlägt der Aufruf MQBEGIN fehl.

Vergewissern Sie sich, dass Ihre Umgebungsvariablen in Ihren Anwendungs- und Warteschlangenmanagerprozessen korrekt definiert sind.

Übersicht über die XA-Aufrufe
Hier finden Sie eine Liste der Aufrufe der XA-Funktionen in einer Datenbankclientbibliothek, die infolge der verschiedenen MQI-Aufrufe zur Steuerung globaler Arbeitseinheiten durchgeführt werden. Hierbei handelt es sich nicht um eine vollständige Beschreibung des in der XA-Spezifikation beschriebenen Protokolls, sondern um eine kurze Übersicht.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Beachten Sie, dass die Aufrufe `xa_start` und `xa_end` immer von WebSphere MQ-Code im Anwendungsprozess aus erfolgen, wohingegen `xa_prepare`, `xa_commit` und `xa_rollback` immer vom Warteschlangenmanageragentenprozess 'amqzlaa0' aufgerufen werden.

Die in dieser Tabelle aufgeführten Aufrufe `xa_open` und `xa_close` erfolgen alle vom Anwendungsprozess aus. Vom Warteschlangenmanageragentenprozess wird `xa_open` unter den im Abschnitt „Fehlerbedingungen" auf Seite 70 beschriebenen Bedingungen aufgerufen.

Tabelle 7. Übersicht über die XA-Funktionsaufrufe

MQI-Aufruf	XA-Aufrufe mit dynamischer Registrierung	XA-Aufrufe ohne dynamische Registrierung
Erster Aufruf MQBEGIN	`xa_open`	`xa_open` `xa_start`

Tabelle 7. Übersicht über die XA-Funktionsaufrufe (Forts.)

MQI-Aufruf	XA-Aufrufe mit dynamischer Registrierung	XA-Aufrufe ohne dynamische Registrierung
Nachfolgende Aufrufe MQBEGIN	Keine XA-Aufrufe	`xa_start`

MQCMIT (**ohne** Aufruf von `ax_reg` während der aktuellen globalen Arbeitseinheit)	Keine XA-Aufrufe	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (**mit** Aufruf von `ax_reg` während der aktuellen globalen Arbeitseinheit)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		Nicht zutreffend. Im nicht dynamischen Modus wird 'ax_reg' nicht aufgerufen.

MQBACK (**ohne** Aufruf von `ax_reg` während der aktuellen globalen Arbeitseinheit)	Keine XA-Aufrufe	`xa_end` `xa_rollback`
MQBACK (**mit** Aufruf von `ax_reg` während der aktuellen globalen Arbeitseinheit)	`xa_end` `xa_rollback`	Nicht zutreffend. Im nicht dynamischen Modus wird 'ax_reg' nicht aufgerufen.
MQDISC, wenn zuerst MQCMIT oder MQBACK aufgerufen wurde. Wurden diese nicht aufgerufen, erfolgt die Verarbeitung von MQCMIT zum ersten Mal während MQDISC.	`xa_close`	`xa_close`

Anmerkungen:

1. Bei MQCMIT wird `xa_commit` aufgerufen, wenn `xa_prepare` erfolgreich ist. Andernfalls wird `xa_rollback` aufgerufen.

Szenario 2: Für die Koordination ist andere Software zuständig

In Szenario 2 werden globale Arbeitseinheiten von einem externen Transaktionsmanager koordiniert, der diese unter der Steuerung der Transaktionsmanager-API startet und festschreibt. Die Verben MQBEGIN, MQCMIT und MQBACK sind nicht verfügbar.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

In diesem Abschnitt wird das Szenario einschließlich folgender Maßnahmen beschrieben:

- „Externe Synchronisationspunktkoordination” auf Seite 73
- „Verwenden von CICS” auf Seite 75
- „Verwenden von Microsoft Transaction Server (COM+)” auf Seite 80

Der IBM WebSphere MQ-Client für HP Integrity NonStop Server kann zur Koordination globaler Arbeitseinheiten die HP NonStop Transaction Management Facility (TMF) verwenden. Weitere Informationen finden Sie im Abschnitt HP NonStop TMF verwenden.

Externe Synchronisationspunktkoordination

Eine globale Arbeitseinheit kann auch von einem externen X/Open XA-kompatiblen Transaktionsmanager koordiniert werden. In diesem Fall ist der WebSphere MQ-Warteschlangenmanager an der Arbeitseinheit beteiligt, koordiniert diese jedoch nicht.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Der Steuerungsfluss in einer von einem externen Transaktionsmanager koordinierten globalen Arbeitseinheit sieht wie folgt aus:

1. Eine Anwendung teilt dem externen Synchronisationspunktkoordinator (z. B. TXSeries) mit, dass sie eine Transaktion starten möchte.
2. Der Synchronisationspunktkoordinator informiert bekannte Ressourcenmanager wie z. B. WebSphere MQ über die aktuelle Transaktion.
3. Die Anwendung gibt Aufrufe an Ressourcenmanager aus, die der aktuellen Transaktion zugeordnet sind. So könnte die Anwendung beispielsweise Aufrufe `MQGET` an WebSphere MQ ausgeben.
4. Die Anwendung setzt eine Festschreibungs- oder Zurücksetzungsanforderung an den externen Synchronisationspunktkoordinator ab.
5. Der Synchronisationspunktkoordinator schließt die Transaktion ab, indem er die entsprechenden Aufrufe an die einzelnen Ressourcenmanager ausgibt, wobei er normalerweise Protokolle für zweiphasige Festschreibung verwendet.

Die unterstützten Versionen externer Synchronisationspunktkoordinatoren, die eine zweiphasige Festschreibung für Transaktionen mit Beteiligung von WebSphere MQ bereitstellen können, sind unter IBM WebSphere MQ -Detaillierte Systemvoraussetzungen definiert.

Im weiteren Teil dieses Abschnitts wird die Vorgehensweise zum Aktivieren externer Arbeitseinheiten beschrieben.

XA-Switchstruktur von IBM WebSphere MQ

Jeder an einer extern koordinierten Arbeitseinheit beteiligte Ressourcenmanager muss eine XA-Switchstruktur bereitstellen. Diese Struktur definiert sowohl die Funktionalität des Ressourcenmanagers als auch die Funktionen, die vom Synchronisationspunktkoordinator aufgerufen werden müssen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

IBM WebSphere MQ stellt zwei Versionen dieser Struktur bereit:

- *`MQRMIXASwitch`* für die statische XA-Ressourcenverwaltung
- *`MQRMIXASwitchDynamic`* für die dynamische XA-Ressourcenverwaltung

Schlagen Sie in der Dokumentation zu Ihrem Transaktionsmanager nach, ob die Schnittstelle für die statische oder die dynamische Ressourcenverwaltung verwendet werden soll. Sofern der Transaktionsmanager dies unterstützt, empfehlen wir die Verwendung der dynamischen XA-Ressourcenverwaltung.

Der Datentyp *long* in der XA-Spezifikation wird von manchen 64-Bit-Transaktionsmanagern als 64-Bit und von manchen als 32-Bit behandelt. WebSphere MQ unterstützt beide Modelle:

- Verwenden Sie bei einem 32-Bit-Transaktionsmanager oder aber bei einem 64-Bit-Transaktionsmanager, der den Datentyp *long* als 32-Bit behandelt, die in Tabelle 8 auf Seite 74 aufgeführte Switchloaddatei.
- Verwenden Sie bei einem 64-Bit-Transaktionsmanager verwenden, der den Datentyp *long* als 64-Bit behandelt, die in Tabelle 9 auf Seite 74 aufgeführte Switchloaddatei.

Eine Liste bekannter 64-Bit-Transaktionsmanager, die den Datentyp *long* als 64-Bit behandeln, finden Sie in Tabelle 10 auf Seite 74. Wenn Sie nicht sicher sind, welches Modell Ihr Transaktionsmanager verwendet, konsultieren Sie die Dokumentation zu Ihrem Transaktionsmanager.

Tabelle 8. Namen von XA-Switchloaddateien		
Plattform	**Name der Switchloaddatei (Server)**	**Name der Switchloaddatei (erweiterter transaktionsorientierter Client)**
Windows	*mqmxa.dll*	*mqcxa.dll*
AIX (ohne Threads)	*libmqmxa.a*	*libmqcxa.a*
AIX (mit Threads)	*libmqmxa_r.a*	*libmqcxa_r.a*
HP-UX (ohne Threads)	*libmqmxa.so*	*libmqcxa.so*
HP-UX (mit Threads)	*libmqmxa_r.so*	*libmqcxa_r.so*
Linux (ohne Threads)	*libmqmxa.so*	*libmqcxa.so*
Linux (Thread)	*libmqmxa_r.so*	*libmqcxa_r.so*
Solaris	*libmqmxa.so*	*libmqcxa.so*

Tabelle 9. Namen alternativer 64-Bit-XA-Switchloaddateien		
Plattform	**Name der Switchloaddatei (Server)**	**Name der Switchloaddatei (erweiterter transaktionsorientierter Client)**
AIX (ohne Threads)	*libmqmxa64.a*	*libmqcxa64.a*
AIX (mit Threads)	*libmqmxa64_r.a*	*libmqcxa64_r.a*
HP-UX (ohne Threads)	*libmqmxa64.so*	*libmqcxa64.so*
HP-UX (mit Threads)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Linux (ohne Threads)	*libmqmxa64.so*	*libmqcxa64.so*
Linux (Thread)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Solaris	*libmqmxa64.so*	*libmqcxa64.so*

Tabelle 10. 64-Bit-Transaktionsmanager, für die die alternative 64-Bit-Switchloaddatei benötigt wird
Transaktionsmanager
Tuxedo

Bei einigen externen Synchronisationspunktkoordinatoren (nicht bei CICS) müssen die einzelnen Ressourcenmanager, die an einer Arbeitseinheit beteiligt sind, ihren Namen im Namensfeld der XA-Switchstruktur angeben. Der Name des WebSphere MQ-Ressourcenmanagers lautet MQSeries_XA_RMI.

Der Synchronisationspunktkoordinator definiert, wie die XA-Switchstruktur von WebSphere MQ eine Verknüpfung mit ihm herstellt. Informationen zur Verknüpfung der XA-Switchstruktur von WebSphere MQ mit CICS finden Sie im Abschnitt „Verwenden von CICS” auf Seite 75. Informationen zur Verknüpfung der XA-Switchstruktur von WebSphere MQ mit anderen XA-kompatiblen Synchronisationspunktkoordinatoren finden Sie in der Dokumentation zu den betreffenden Produkten.

Die folgenden Überlegungen gelten für die Verwendung von WebSphere MQ mit allen XA-kompatiblen Synchronisationspunktkoordinatoren:

- In der in einem Aufruf 'xa_open' vom Synchronisationspunktkoordinator übergebenen Struktur 'xa_info' ist der Name eines WebSphere MQ-Warteschlangenmanagers enthalten. Der Name hat dasselbe For-

mat wie der an den Aufruf MQCONN übergebene Warteschlangenmanagername. Falls im Aufruf 'xa_open' kein Name übergeben wird, wird der Standardwarteschlangenmanager verwendet.

Alternativ dazu können in der Struktur 'xa_info' Werte für die Parameter *TPM* und *AXLIB* enthalten sein. Mit dem Parameter *TPM* wird der verwendete Transaktionsmanager angegeben. Gültig sind die Werte CICS, TUXEDO und ENCINA. Der Parameter *AXLIB* gibt den Namen der Bibliothek an, in der die Funktionen 'ax_reg' und 'ax_unreg' des Transaktionsmanagers enthalten sind. Weitere Informationen zu diesen Parametern finden Sie im Abschnitt Erweiterten transaktionsorientierten Client konfigurieren. Falls einer dieser Parameter in der Struktur 'xa_info' enthalten ist, wird der Warteschlangenmanagername im Parameter *QMNAME* angegeben, sofern nicht der Standardwarteschlangenmanager verwendet wird.

- An einer Transaktion, die von einer Instanz eines externen Synchronisationspunktkoordinators koordiniert wird, kann immer nur jeweils ein Warteschlangenmanager beteiligt sein. Der Synchronisationspunktkoordinator wird mit dem Warteschlangenmanager verbunden und für ihn gilt die Regel, dass nur jeweils eine Verbindung unterstützt wird.
- Alle Anwendungen, die Aufrufe an einen externen Synchronisationspunktkoordinator umfassen, können nur zu dem Warteschlangenmanager eine Verbindung herstellen, der an der vom externen Koordinator verwalteten Transaktion beteiligt ist (da sie bereits mit diesem Warteschlangenmanager verbunden sind). Allerdings müssen diese Anwendungen einen Aufruf MQCONN zur Anforderung einer Verbindungskennung und einen Aufruf MQDISC ausgeben, bevor sie beendet werden.
- Ein Warteschlangenmanager mit Ressourcenaktualisierungen, die von einem externen Synchronisationspunktkoordinator koordiniert werden, muss vor dem externen Synchronisationspunktkoordinator gestartet werden. Ebenso muss der Synchronisationspunktkoordinator vor dem Warteschlangenmanager beendet werden.
- Wenn der externe Synchronisationspunktkoordinator abnormal beendet wird, stoppen und starten Sie den Warteschlangenmanager erneut, ***bevor*** Sie den Synchronisationspunktkoordinator erneut starten, um sicherzustellen, dass alle Messagingoperationen, die zum Zeitpunkt des Fehlers nicht festgeschrieben waren, ordnungsgemäß aufgelöst werden.

Verwenden von CICS

CICS ist eines der Elemente von TXSeries.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Die Versionen von TXSeries , die XA-konform sind (und einen zweiphasigen Festschreibungsprozess verwenden), sind unter IBM WebSphere MQ -Detaillierte Systemvoraussetzungen definiert.

Auch andere Transaktionsmanager werden von WebSphere MQ unterstützt. Im Abschnitt IBM WebSphere MQ -Detaillierte Systemvoraussetzungen finden Sie die aktuellen Listen der unterstützten Software.

Anforderungen der zweiphasigen Festschreibung

Die Anforderungen der zweiphasigen Festschreibung bei Verwendung der zweiphasigen Festschreibung von CICS mit WebSphere MQ. Diese Anforderungen gelten nicht für z/OS.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Beachten Sie die folgenden Anforderungen:

- WebSphere MQ und CICS müssen sich auf derselben physischen Maschine befinden.
- WebSphere MQ unterstützt CICS nicht auf einem WebSphere MQ MQI-Client.
- Sie müssen den Warteschlangenmanager starten und dabei seinen Name in der Zeilengruppe für die XAD-Ressourcendefinition angeben, ***bevor*** Sie versuchen, CICS zu starten. Wenn Sie dies nicht tun, können Sie CICS nicht starten, wenn Sie eine Zeilengruppe zur XAD-Ressourcendefinition für WebSphere MQ zur CICS-Region hinzugefügt haben.

- Es kann immer nur auf einen WebSphere MQ-Warteschlangenmanager von einer einzelnen CICS-Region zugegriffen werden.
- Eine CICS-Transaktion kann erst nach Ausgabe einer Anforderung `MQCONN` auf WebSphere MQ-Ressourcen zugreifen. Im Aufruf `MQCONN` muss der Name des WebSphere MQ-Warteschlangenmanagers angegeben sein, der im XAOpen-Eintrag der Zeilengruppe zur XAD-Ressourcendefinition für die CICS-Region angegeben ist. Ist dieser Eintrag leer, muss in der Anforderung `MQCONN` der Standardwarteschlangenmanager angegeben sein.
- Eine CICS-Transaktion, die auf WebSphere MQ-Ressourcen zugreift, muss von der Transaktion aus einen Aufruf `MQDISC` ausgeben, bevor sie zu CICS zurückkehrt. Wenn dies nicht geschieht, ist der CICS-Anwendungsserver möglicherweise weiterhin verbunden und es bleiben Warteschlangen geöffnet. Wenn Sie keinen Exit für die Taskbeendigung installieren (siehe Abschnitt „Beispiel für einen Taskbeendigungsexit” auf Seite 79) kann es darüber hinaus passieren, dass der CICS-Anwendungsserver später bei einer nachfolgenden Transaktion abnormal beendet wird.
- Sie müssen sicherstellen, dass die CICS-Benutzer-ID (cics) Mitglied der Gruppe 'mqm' ist, sodass der CICS-Code zum Aufruf von WebSphere MQ berechtigt ist.

  Bei Transaktionen, die in einer CICS-Umgebung ausgeführt werden, passt der Warteschlangenmanager seine Berechtigungsmethoden und den entscheidenden Kontext wie folgt an:

  – Der Warteschlangenmanager fragt die Benutzer-ID, unter der CICS die Transaktion ausführt, ab. Hierbei handelt es sich um die Benutzer-ID, die vom Objektberechtigungsmanager geprüft und für Kontextinformationen verwendet wird.
  – Im Nachrichtenkontext lautet der Anwendungstyp MQAT_CICS.
  – Der Anwendungsname in dem Kontext wird aus dem CICS-Transaktionsnamen kopiert.

Allgemeine XA-Unterstützung
Allgemeine XA-Unterstützung wird unter IBM inicht unterstützt. Über ein bereitgestelltes XA-Switchloadmodul können Sie CICS mit WebSphere MQ auf UNIX and Linux-Systemen verknüpfen. Darüber hinaus sind Beispielquellcodedateien verfügbar, mit deren Hilfe Sie die XA-Switches für andere Transaktionsnachrichten entwickeln können.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Die Namen der bereitgestellten Switchloadmodule:

Tabelle 11. Essenzieller Code für CICS-Anwendungen: XA-Initialisierungsroutine

C (Quelle)	**C (Exec) - fügen Sie eine der folgenden Komponenten zu Ihrer XAD.Stanza**
amqzscix.c	amqzsc - TXSeries für AIXVersion 5.1, amqzsc - TXSeries für HP-UXVersion 5.1 amqzsc - TXSeries für Sun Solaris Version 5.1
amqzscin.c	mqmc4swi - TXSeries für WindowsVersion 5.1

Bibliotheken zur Verwendung in TXSeries for Multiplatforms erstellen
Dieser Abschnitt enthält Informationen zum Erstellen von Bibliotheken, die in TXSeries for Multiplatforms verwendet werden sollen.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Bei *vordefinierten Switchloaddateien* handelt es sich um gemeinsam genutzte Bibliotheken (*DLLs* unter Windows) für die Verwendung mit CICS-Programmen, in denen eine 2-Phasen-Festschreibungstransaktion mit dem XA-Protokoll erforderlich ist. Die Namen dieser vordefinierten Bibliotheken sind in der Tabelle Wesentlicher Code für CICS-Anwendungen: XA-Initialisierungsroutine aufgeführt. Beispielquellcode ist außerdem in den folgenden Verzeichnissen enthalten:

Tabelle 12. Installationsverzeichnisse unter Windows und UNIX and Linux

Plattform	**Directory**	**Quellendatei**
UNIX and Linux	*MQ_INSTALLATION_PATH*/ samp/	amqzscix.c
Windows	*MQ_INSTALLATI-ON_PATH*\Tools\C\Samples	amqzscin.c

Dabei steht*MQ_INSTALLATION_PATH* für das Verzeichnis, in dem Sie IBM WebSphere MQinstalliert haben.

Befolgen Sie die entsprechenden Anweisungen für Ihr Betriebssystem, um die Switchloaddatei aus der Beispielquelle zu erstellen:

AIX

Geben Sie den folgenden Befehl ein:

```
export MQM_HOME=/usr/mqm
echo "amqzscix" > tmp.exp
xlc_r $MQM_HOME/samp/amqzscix.c -I/usr/lpp/cics/include -I$MQM_HOME/inc -e amqzscix -bE:tmp.exp -bM:SRE
-o amqzsc /usr/lpp/cics/lib/regxa_swxa.o -L$MQM_HOME/lib -L/usr/lpp/cics/lib -lcicsrt -lEncina -lEncSer
ver -lpthreads -lsarpc -lmqmcics_r -lmqmxa_r -lmqzi_r -lmqmcs_r
rm tmp.exp
```

Solaris

Geben Sie den folgenden Befehl ein:

```
/opt/SUNWspro/bin/cc -s -l/opt/encina/include amqzscix.c -G -o amqzscix  -e
    CICS_XA_Init -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib
    -L/opt/dcelocal/lib /opt/cics/lib/reqxa_swxa.o
    -lmqmcics -lmqmxa -lmqzi -lmqmcs -lmqmzse -lcicsrt -lEncina -lEncSfs -ldce
```

HP-UX

Geben Sie den folgenden Befehl ein:

```
cc -c -s -I/opt/encina/include MQ_INSTALLATION_PATH/samp/amqzscix.c -Aa +z -o  amqzscix.o ld -b
    -o amqzscix amqzscix.o /opt/cics/lib/regxa_swxa.o +e CICS_XA_Init \
    -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib -L/opt/cics/lib
    -lmqmxa_r -lmqzi_r -lmqmcs_r -lmqmzse -ldbm -lc -lm
```

Linux-Plattformen

Geben Sie den folgenden Befehl ein:

```
gcc -m32 -shared -fPIC -o amqzscix amqzscix.c
    \-IMQ_INSTALLATION_PATH/inc -I CICS_INSTALLATION_PATH/include
    \-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
    \-Wl,-rpath=/usr/lib -Wl,-rpath-link,/usr/lib -Wl,--no-undefined
    -Wl,--allow-shlib-undefined \-L CICS_LIB_PATH/regxa_swxa.o \-lpthread -ldl -lc
    -shared -lmqzi_r -lmqmxa_r -lmqmcics_r -ldl -lc
```

Windows

Führen Sie folgende Schritte aus:

1. Verwenden Sie den Befehl 'cl', um 'amqzscin.obj' durch Kompilation zu erstellen. Geben Sie dabei mindestens Folgendes an:

```
cl.exe -c -IEncinaPath\include -IMQ_INSTALLATION_PATH\include -Gz -LD amqzscin.c
```

2. Erstellen Sie die Moduldefinitionsdatei `mqmc1415.def` mit den folgenden Zeilen:

```
LIBRARY MQMC4SWI
EXPORTS
CICS_XA_Init
```

3. Verwenden Sie den Befehl **lib**, um eine Exportdatei und eine Importbibliothek zu erstellen. Geben Sie dabei mindestens die folgenden Optionen an:

```
lib -def:mqmc4swi.def -out:mqmc4swi.lib
```

   Bei erfolgreicher Ausführung des Befehls 'lib' wird auch die Datei `mqmc4swi.exp` erstellt.

4. Verwenden Sie den Befehl 'link', um die Datei `mqmc4swi.dll` zu erstellen. Geben Sie dabei mindestens die folgende Option an:

```
link.exe -dll -nod -out:mqmc4swi.dll
    amqzscin.obj CicsPath\lib\regxa_swxa.obj
    mqmc4swi.exp mqmcics4.lib
   CicsPath\lib\libcicsrt.lib
   DcePath\lib\libdce.lib DcePath\lib\pthreads.lib
   EncinaPath\lib\libEncina.lib
   EncinaPath\lib\libEncServer.lib
   msvcrt.lib kernel32.lib
```

IBM WebSphere MQ-XA-Unterstützung und Tuxedo
IBM WebSphere MQ Unter WindowskönnenUNIX and Linux -Systeme Tuxedo-koordinierte XA-Anwendungen unbegrenzt in xa_start blockieren.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Dies kann nur passieren, wenn zwei oder mehr Prozesse, die von Tuxedo in einer einzelnen globalen Transaktion koordiniert werden, versuchen, unter Verwendung derselben Transaktionsverzweigungs-ID (XID) auf IBM WebSphere MQ zuzugreifen. Wenn jeder Prozess in der globalen Transaktion von Tuxedo eine andere XID zur Verwendung mit IBM WebSphere MQ erhält, kann diese Situation nicht auftreten.

Konfigurieren Sie zur Vermeidung des Problems jede Anwendung in Tuxedo, die auf IBM WebSphere MQ unter einer einzelnen globalen Transaktions-ID (gtrid) zugreift, innerhalb ihrer eigenen Tuxedo-Servergruppe. Prozesse in derselben Servergruppe verwenden beim Zugriff auf Ressourcenmanager im Auftrag einer einzelnen gtrid dieselbe XID, es besteht daher die Gefahr, dass sie bei xa_start in IBM WebSphere MQ blockiert werden. Prozesse in verschiedenen Servergruppen verwenden beim Zugriff auf Ressourcenmanager separate XIDs und müssen daher ihre Transaktionsarbeit in IBM WebSphere MQ nicht serialisieren.

Zweiphasige Festschreibung von CICS aktivieren
Damit CICS zur Koordinierung von Transaktionen mit MQI-Aufrufen eine zweiphasige Festschreibung verwenden kann, fügen Sie einen CICS-Zeilengruppeneintrag für die XAD-Ressourcendefinition zur CICS-Region hinzu. Die Informationen in diesem Abschnitt gelten nicht für z/OS.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Im folgenden Beispiel wird ein XAD-Zeilengruppeneintrag für WebSphere MQ für Windowshinzugefügt, wobei <Drive> das Laufwerk ist, auf dem WebSphere MQ installiert ist (z. B. D:).

```
cicsadd -cxad -r<cics_region> \
     ResourceDescription="MQM XA Product Description" \
     SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
     XAOpen=<queue_manager_name>
```

Verwenden Sie für erweiterte transaktionsorientierte Clients die Switchloaddatei 'mqcc4swi.dll'.

Das folgende Beispiel veranschaulicht, wie ein XAD-Zeilengruppeneintrag für WebSphere MQ auf Systemen mit UNIX and Linux hinzugefügt wird; dabei ist *MQ_INSTALLATION_PATH* das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist:

```
cicsadd -cxad -r<cics_region> \
      ResourceDescription="MQM XA Product Description" \
      SwitchLoadFile="MQ_INSTALLATION_PATH/lib/amqzsc" \
      XAOpen=<queue_manager_name>
```

Verwenden Sie für erweiterte transaktionsorientierte Clients die Switchloaddatei 'amqczsc'.

Informationen zur Verwendung des Befehls **cicsadd** finden Sie in der *CICS Administration Reference* oder im *CICS Administration Guide* für Ihre Plattform.

Aufrufe an WebSphere MQ können in eine CICS-Transaktion aufgenommen werden und die WebSphere MQ-Ressourcen werden gemäß CICS-Anweisung festgeschrieben oder rückgängig gemacht. Diese Unterstützung ist für Clientanwendungen nicht verfügbar.

Sie ***müssen*** eine MQCONN aus Ihrer CICS -Transaktion ausgeben, um auf WebSphere MQ -Ressourcen zuzugreifen, gefolgt von einem entsprechenden MQDISC beim Beenden.

CICS-Benutzerexits aktivieren
Ein CICS-Benutzerexit-*Punkt* (normalerweise als *Benutzerexit* bezeichnet) ist eine Stelle in einem CICS-Modul, an der CICS die Kontrolle an ein von Ihnen geschriebenes Programm (ein Benutzerexit-*Programm*) übertragen kann und an dem CICS die Kontrolle wieder übernehmen kann, nachdem Ihr Exitprogramm seine Arbeit abgeschlossen hat.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Lesen Sie vor der Verwendung eines CICS-Benutzerexits den *CICS Administration Guide* für Ihre Plattform.

Beispiel für einen Taskbeendigungsexit
In WebSphere MQ ist Beispielquellcode für einen CICS-Taskbeendigungsexit verfügbar.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Der Beispielquellcode befindet sich in den folgenden Verzeichnissen:

Tabelle 13. CICS-Taskbeendigungsexits

Plattform	**Directory**	**Quellendatei**
UNIX and Linux-Systeme	*MQ_INSTALLATION_PATH*/samp	amqzscgx.c
Windows	*MQ_INSTALLATI-ON_PATH*\Tools\C\Samples	amqzscgn.c

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

Die Buildanweisungen für den Beispieltaskbeendigungsexit sind in den Kommentaren nahe dem Anfang jeder Quellendatei enthalten.

Dieser Exit wird von CICS bei der normalen und der abnormalen Taskbeendigung (nach einem Synchronisationspunkt) aufgerufen. Im Exitprogramm ist keine wiederherstellbare Arbeit zulässig.

Diese Funktionen werden nur in einem WebSphere MQ- und CICS-Kontext verwendet, in dem die CICS-Version die XA-Schnittstelle unterstützt. In CICS werden diese Bibliotheken als "Programme" oder "Benutzerexits" bezeichnet.

In CICS gibt es eine Reihe von Benutzerexits, und falls amqzscgx verwendet wird, wird es in CICS als "Task termination user exit (UE014015)", also Exitnummer 15, definiert und aktiviert.

Wenn der Taskbeendigungsexit von CICS aufgerufen wird, hat CICS WebSphere MQ bereits über den Beendigungsstatus der Task informiert und WebSphere MQ hat die entsprechende Maßnahme (Commit oder Rollback) ergriffen. Der Exit gibt lediglich einen Aufruf MQDISC für eine Bereinigung aus.

Ein Zweck der Installation und Konfiguration des CICS-Systems für die Verwendung eines Taskbeendigungsexits besteht darin, das System vor einigen Konsequenzen fehlerhaften Anwendungscodes zu schützen. Wenn beispielsweise die CICS-Transaktion abnormal beendet wird, ohne zuerst MQDISC aufzurufen, und kein Taskbeendigungsexit installiert ist, wird möglicherweise (innerhalb von ca. 10 Sekunden) ein nachfolgender nicht behebbarer Fehler der CICS-Region angezeigt. Dies liegt daran, dass der WebSphere MQ-Thread für den fehlerfreien Zustand, der in dem cicsas-Prozess ausgeführt wird, nicht gesendet wurde und keine Zeit zur Bereinigung und Rückgabe hatte. Dies könnte sich daran zeigen, dass der cicsas-Prozess umgehend beendet wird, nachdem er FFST-Berichte in das Verzeichnis '/var/mqm/errors' oder ein äquivalentes Verzeichnis unter Windows geschrieben hat.

Verwenden von Microsoft Transaction Server (COM+)

COM + (Microsoft Transaction Server) unterstützt Benutzer bei der Ausführung von Geschäftslogikanwendungen auf einem typischen Server der Mittelschicht.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Wichtige Informationen finden Sie unter Features, die nur mit der primären Installation unter Windows verwendet werden können .

COM + unterteilt die Arbeit in *Aktivitäten*, bei denen es sich in der Regel um kurze unabhängige Geschäftslogikblöcke handelt, z. B. *Transferfonds von Konto A auf Konto B*. COM + stützt sich stark auf Objektorientierung und insbesondere auf COM; lose wird eine COM + -Aktivität wird durch ein COM (business) -Objekt repräsentiert.

COM+ ist im Betriebssystem integriert. Soll COM+ unter Windows 2000 und Windows XP verwendet werden, benötigen Sie Hotfix Q313582 (auch unter dem Namen 'COM+ Rollup Package 19.1' bekannt).

Mit COM+ erhält der Geschäftsobjektadministrator drei Services, die drei für den Programmierer von Geschäftsobjekten wichtige Bereiche abdecken:

- Transaktionsmanagement
- Sicherheit
- Ressourcenpool

COM+ wird in der Regel mit Front-End-Code verwendet, bei dem es sich um einen COM-Client für die in COM+ enthaltenen Objekte handelt, und mit Back-End-Services wie beispielsweise einer Datenbank, wobei WebSphere MQ die Brücke zwischen dem COM+-Geschäftsobjekt und dem Back-End darstellt.

Der Front-End-Code kann ein Standalone-Programm oder eine Active Server Page (ASP) auf dem Microsoft Internet Information Server (IIS) sein. Dieser Front-End-Code kann sich auf demselben Computer wie COM+ und die Geschäftsobjekte befinden, wobei die Verbindung über COM erfolgt. Der Front-End-Code kann sich auch auf einem anderen Computer befinden; in diesem Fall erfolgt die Verbindung über DCOM. Sie können mehrere Clients verwenden, die in unterschiedlichen Situationen jeweils auf dasselbe COM+-Geschäftsobjekt zugreifen.

Der Back-End-Code kann sich auf demselben Computer wie COM+ und die Geschäftsobjekte befinden oder auf einem anderen Computer; dabei erfolgt die Verbindung über eines der von WebSphere MQ unterstützten Protokolle.

Globale Arbeitseinheiten beenden

Der Warteschlangenmanager kann für das Beenden globaler Arbeitseinheiten nach einem vorkonfigurierten Inaktivitätsintervall konfiguriert werden.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

Legen Sie die folgenden Umgebungsvariablen fest, um diese Option zu aktivieren:

- *AMQ_TRANSACTION_EXPIRY_RESCAN*= < Rescan-Intervall in Millisekunden >
- *AMQ_XA_TRANSACTION_EXPIRY*= < Zeitlimitintervall in Millisekunden >

Achtung: Die Umgebungsvariablen wirken sich nur auf Transaktionen aus, die in der Tabelle 6-4 der XA-Spezifikation den Status *Idle* (Inaktiv) haben. Dies betrifft also nur Transaktionen, die in keinem Anwendungsthread zugeordnet sind und für die der externe Transaktionsmanager noch keinen **`xa_prepare`**-Funktionsaufruf ausgegeben hat.

Externe Transaktionsmanager bewahren ausschließlich Protokolle von Transaktionen auf, die vorbereitet, festgeschrieben oder rückgängig gemacht wurden. Wenn der externe Transaktionsmanager aus irgendeinem Grund beendet wird, bringt er nach seiner Rückkehr vorbereitete, festgeschriebene und rückgängig gemachte Transaktionen zum Abschluss; aktive, noch nicht vorbereitete Transaktionen verwaisen jedoch. Legen Sie *AMQ_XA_TRANSACTION_EXPIRY* unter Berücksichtigung des erwarteten Intervalls zwischen der Durchführung von transaktionsorientierten MQI-API-Aufrufen durch eine Anwendung und dem Abschluss der Transaktion nach Durchführung transaktionsorientierter Arbeitsvorgänge auf anderen Ressourcenmanagern fest, um dies zu vermeiden.

Um eine rechtzeitige Bereinigung nach Ablauf von *AMQ_XA_TRANSACTION_EXPIRY* zu gewährleisten, muss *AMQ_TRANSACTION_EXPIRY_RESCAN* auf einen niedrigeren Wert als das Intervall *AMQ_XA_TRANSACTION_EXPIRY* gesetzt werden, idealerweise so, dass der Neuscan mehrmals während des Intervalls *AMQ_XA_TRANSACTION_EXPIRY* erfolgt.

Disposition der Arbeitseinheit mit Wiederherstellung

WebSphere MQ für z/OS bietet die Möglichkeit der Disposition von Arbeitseinheiten mit Wiederherstellung. Mit dieser Funktion können Sie konfigurieren, ob die zweite Phase von Transaktionen mit zweiphasiger Festschreibung beispielsweise während der Wiederherstellung gesteuert werden kann, wenn eine Verbindung zu einem anderen Warteschlangenmanager innerhalb derselben Gruppe mit gemeinsamer Warteschlange (QSG) besteht.

Anmerkung: Dieses Thema ist auch in IBM MQ Version 8.0 und höheren Versionen verfügbar. Sie können jedoch nicht über das Listenfeld "Version ändern" zu einer späteren Version wechseln. Um das Thema in einer späteren Version aufzurufen, bearbeiten Sie die Versionsnummer im URL-Feld in Ihrem Browser.

WebSphere MQ für z/OS V7.0.1 und höher unterstützt die Disposition von Arbeitseinheiten mit Wiederherstellung.

Disposition der Arbeitseinheit mit Wiederherstellung

Die Disposition von Arbeitseinheiten mit Wiederherstellung bezieht sich auf die Verbindung einer Anwendung und nachfolgend auf alle von dieser gestarteten Transaktionen. Es gibt zwei mögliche Dispositionen von Arbeitseinheiten mit Wiederherstellung.

- Bei einer Disposition GROUP der Arbeitseinheit mit Wiederherstellung ist eine transaktionsorientierte Anwendung logisch mit der Gruppe mit gemeinsamer Warteschlange verbunden und hat keine Affinität zu einem bestimmten Warteschlangenmanager. Alle Transaktionen mit zweiphasiger Festschreibung, die von dieser gestartet werden und die Phase 1 des Festschreibungsprozesses abgeschlossen haben, also unbestätigt sind, können abgefragt und aufgelöst werden, wenn sie mit einem Warteschlangenmanager in der QSG verbunden sind. In einem Wiederherstellungsszenario bedeutet dies, dass der Transaktionskoordinator die Verbindung zu demselben Warteschlangenmanager, der möglicherweise nicht mehr verfügbar ist, nicht wiederherstellen muss.

- Bei einer Disposition QMGR der Arbeitseinheit mit Wiederherstellung hat eine Anwendung direkte Affinität zu dem Warteschlangenmanager, mit dem sie verbunden ist, und auch alle von ihr gestarteten Anwendungen haben ebenfalls diese Disposition.

  In einem Wiederherstellungsszenario muss der Transaktionskoordinator die Verbindung zu demselben Warteschlangenmanager wiederherstellen, um unbestätigte Transaktionen abzufragen und aufzulösen. Dies gilt unabhängig davon, ob der Warteschlangenmanager zu einer Gruppe mit gemeinsamer Warteschlange gehört oder nicht.

Entscheiden, welche Programmiersprache verwendet werden soll

Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

IBM WebSphere MQ unterstützt die folgenden prozeduralen Programmiersprachen:

- C
- Visual Basic (nur Windows -Systeme)
- COBOL

Diese Sprachen verwenden die Schnittstelle für Nachrichtenwarteschlangen (MQI; Message Queue Interface) für den Zugriff auf Message-Queueing-Services. Weitere Information über Unterstützung für diese Sprachen finden Sie unter „Prozedurale Programmiersprachen mit WebSphere MQ verwenden” auf Seite 82.

IBM WebSphere MQ unterstützt:

- .NET
- ActiveX
- C + +
- Java
- JMS

Diese Sprachen verwenden das IBM WebSphere MQ-Objektmodell, das Klassen umfasst, die dieselbe Funktionalität wie WebSphere MQ-Aufrufe und -Strukturen bereitstellen, jedoch eine natürlichere Art der Programmierung in einer objektorientierten Umgebung sind. Einige der Sprachen, die das IBM WebSphere MQ-Objektmodell verwenden, bieten zusätzliche Funktionen, die in der Schnittstelle für Nachrichtenwarteschlangen (Message Queue Interface; MQI) nicht verfügbar sind. Weitere Information über Unterstützung für diese Sprachen finden Sie unter „Objektorientierte Programmierung mit WebSphere MQ” auf Seite 83.

Prozedurale Programmiersprachen mit WebSphere MQ verwenden

Weitere Informationen zum Schreiben von Anwendungen in der von Ihnen gewünschten Programmiersprache finden Sie unter nachstehenden Links:

- „Codierung in C” auf Seite 86
- „Codierung in Visual Basic” auf Seite 91
- „Codierung in COBOL” auf Seite 89

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.

WebSphere MQ stellt Datendefinitionsdateien zur Verfügung, um Sie beim Schreiben von Anwendungen zu unterstützen. Eine ausführliche Beschreibung finden Sie unter „IBM WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

Wenn Sie wählen können, in welcher Sprache Sie Programme codieren, bedenken Sie die maximale Länge der Nachrichten, die die Programme verarbeiten werden. Sofern die Programme nur Nachrichten mit einer bekannten maximalen Länge verarbeiten, können Sie sie in allen unterstützen Programmiersprachen codieren. Ist Ihnen die maximale Länge der Nachrichten, die die Programme verarbeiten werden müssen, jedoch nicht bekannt, hängt die Auswahl der Sprache davon ab, ob Sie eine CICS-, IMS- oder Stapelanwendung schreiben:

IMS und Stapel
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 jeder beliebigen von CICS unterstützten Sprache. Die EXEC CICS-Schnittstelle stellt ggf. die Aufrufe für das Speichermanagement zur Verfügung.

Objektorientierte Programmierung mit WebSphere MQ

Einige der Sprachen und Programmierungsrahmendefinitionen, die das IBM WebSphere MQ-Objektmodell verwenden, bieten zusätzliche Funktionen, die in der Schnittstelle für Nachrichtenwarteschlangen (Message Queue Interface; MQI) nicht verfügbar sind. Weitere Informationen zu den Klassen, Methoden und Eigenschaften des IBM WebSphere MQ-Objektmodells erhalten Sie im Abschnitt „Das IBM WebSphere MQ-Objektmodell” auf Seite 91.

.NET
Unter Verwendung von .NET erhalten Sie Informationen zur Codierung von .NET-Programmen unter Verwendung der WebSphere MQ .NET-Klassen. Message Service Clients für C/C++ und .NET stellen eine Anwendungsprogrammierschnittstelle (API) mit der Bezeichnung XMS zur Verfügung, die über dieselben Schnittstellen wie die JMS-API (Java Message Service) verfügt.

C + +
IBM WebSphere MQ bietet C++-Klassen, die WebSphere MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die den Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen. Informationen zum Codieren von Programmen unter Verwendung des WebSphere MQ -Objektmodells in C++ finden Sie unter Verwendung von C++ . Message Service Clients for C/C++ and .NET stellen eine Anwendungsprogrammierschnittstelle (API) mit dem Namen XMS bereit, die über dieselbe Gruppe von Schnittstellen wie der Java Message Service (JMS) verfügt. API.

Java
Informationen zum Codieren von Programmen mit dem WebSphere MQ -Objektmodell in Java finden Sie unter Java verwenden . Weitere Informationen zu den Unterschieden zwischen IBM WebSphere MQ Classes for Java und IBM WebSphere MQ-Klassen, um Ihnen die Entscheidung zu erleichtern, welche Klassen am besten verwendet werden sollten, erhalten Sie unter „Soll ich IBM WebSphere MQ Classes for Java oder IBM WebSphere MQ Classes for JMS verwenden?” auf Seite 93.

JMS
WebSphere MQ stellt darüber hinaus Klassen zur Verfügung, die die JMS-Spezifikation (Java Message Service) implementieren. Weitere Informationen über die WebSphere MQ-Klassen für Java Message Service erhalten Sie unter Java verwenden. Weitere Informationen zu den Unterschieden zwischen IBM WebSphere MQ Classes for Java und IBM WebSphere MQ-Klassen, um Ihnen die Entscheidung zu erleichtern, welche Klassen am besten verwendet werden sollten, erhalten Sie unter „Soll ich IBM WebSphere MQ Classes for Java oder IBM WebSphere MQ Classes for JMS verwenden?” auf Seite 93.

Message Service Clients für C/C++ und .NET stellen eine Anwendungsprogrammierschnittstelle (API) mit der Bezeichnung XMS zur Verfügung, die über dieselben Schnittstellen wie die JMS-API (Java Message Service) verfügt.

ActiveX
WebSphere MQ ActiveX ist allgemein als MQAX bekannt. MQAX ist Bestandteil von WebSphere MQ for Windows. Unterstützung für ActiveX wurde in WebSphere MQ Version 6.0 stabilisiert. Wenn Sie die

Funktionen für WebSphere MQ ab Version 6.0 nutzen möchten, sollten Sie besser .NET verwenden. Unter Verwendung der Component Object Model-Schnittstelle (WebSphere MQ Automation Classes for ActiveX) erhalten Sie Informationen über die Codierung von Programmen unter Verwendung des WebSphere MQ-Objektmodells in ActiveX.

Zugehörige Konzepte

Technische Übersicht

„Anwendungen entwickeln” auf Seite 7
IBM WebSphere MQ bietet mehrere Möglichkeiten, mit denen Sie Anwendungen entwickeln können, um Nachrichten zur Unterstützung Ihrer Geschäftsprozesse zu senden und zu empfangen. Darüber hinaus können Sie Anwendungen für das Management Ihrer Warteschlangenmanager und zugehörigen Ressourcen entwickeln.

„Konzepte für die Anwendungsentwicklung” auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

Zugehörige Verweise

Verweis auf die Anwendungsentwicklung

IBM WebSphere MQ-Datendefinitionsdateien

IBM WebSphere 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 behilflich sein sollen, werden im Abschnitt Kopierdateien, Headerdateien, einzuschließende Dateien und Moduldateien von WebSphere MQ beschrieben.

Die Datendefinitionsdateien, die Ihnen beim Schreiben von Exits für installierbare Services helfen sollen, werden im Abschnitt „Benutzerexits, API-Exits und installierbare WebSphere MQ-Services” auf Seite 398 beschrieben.

Informationen zu Datendefinitionsdateien, die von C++ unterstützt werden, finden Sie unter Verwendung von C++.

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

Zusatz	Sprache
v	COBOL (mit gesetzten Standardwerten)

Installationsbibliothek

Der Name **thlqual** ist das übergeordnete Qualifikationsmerkmal der Installationsbibliothek unter z/OS.

Dieser Abschnitt enthält eine Einführung zu WebSphere MQ-Datendefinitionsdateien unter folgenden Überschriften:

- „Include-Dateien in Programmiersprache C” auf Seite 85
- „Visual Basic-Moduldateien” auf Seite 85
- „Kopierdateien in COBOL” auf Seite 85

Include-Dateien in Programmiersprache C

Die C-Kopfdatendateien von WebSphere MQ sind im Abschnitt C-Kopfdatendateien aufgeführt. Sie werden in folgenden Verzeichnissen oder Bibliotheken installiert:

Plattform	Installationsverzeichnis oder Bibliothek
UNIX-Plattformen	*MQ_INSTALLATION_PATH*/inc/
Windows-Systeme	*MQ_INSTALLATION_PATH*\Tools\c\include

Dabei steht *MQ_INSTALLATION_PATH* für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.

Anmerkung: Auf UNIX-Plattformen werden die Kopfdatendateien symbolisch im Verzeichnis `/usr/include` verknüpft.

Weitere Informationen zur Struktur von Verzeichnissen finden Sie unter Dateisystemunterstützung planen .

Visual Basic-Moduldateien

WebSphere MQ for Windows stellt vier Visual Basic-Moduldateien 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 WebSphere MQ separate Kopierdateien, die die benannten Konstanten enthalten, und zwei Kopierdateien für jede der 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.

Die COBOL-Kopierdateien von WebSphere sind im Abschnitt COBOL-Kopierdateien aufgelistet. Sie werden in folgenden Verzeichnissen installiert:

Plattform	Installationsverzeichnis oder Bibliothek
Sonstige UNIX-Plattformen	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (für Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (für IBM VisualAge COBOL)

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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.

Codierung in C

Beachten Sie bei der Codierung von WebSphere MQ-Programmen in der Programmiersprache C die Informationen in den folgenden Abschnitten.

- „Parameter der MQI-Aufrufe” auf Seite 87
- „Parameter mit einem nicht definierten Datentyp” auf Seite 87
- „Datentypen” auf Seite 87

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
```

Codierung in COBOL

Beachten Sie bei der Codierung von WebSphere MQ-Programmen in der Programmiersprache 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 WebSphere 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 WebSphere MQ-Strukturen gemeinsam, damit sie alle an Wortgrenzen ausgerichtet sind.

Codierung in pTAL

Beachten Sie bei der Codierung von IBM WebSphere MQ-Programmen in pTAL die Informationen im folgenden Abschnitt.

HP Integrity NonStop Server

IBM WebSphere MQ-Strukturen definieren und initialisieren

Es stehen pTAL-Strukturdefinitionen für die IBM WebSphere MQ-Strukturen zur Verfügung. Die Namen dieser Definitionen enden mit ^DEF. So würden beispielsweise die folgenden pTAL-Deklarationen codiert, um eine IBM WebSphere MQ-Nachrichtendeskriptorstruktur (MQMD) und eine IBM WebSphere MQ-Struktur zum Einreihen von Nachrichten (MQPMO) zu erstellen.

```
    STRUCT MYMD(MQMD^DEF);        ! Declare an MQMD structure
    STRUCT MYPMO(MQPMO^DEF);      ! Declare an MQPMO structure
```

In IBM WebSphere MQ gibt es pTAL DEFINE mit Namen, die mit ^DEFAULT enden. Damit können die IBM WebSphere MQ-Strukturen mit Standardwerten initialisiert werden. Mit den folgenden pTAL-Anweisungen sollen den deklarierten Strukturen MQMD und MQPMO Standardwerte zugewiesen werden:

```
    MQMD^DEFAULT(MYMD);           ! Assign default values to an MQMD structure
    MQPMO^DEFAULT(MYPMO);         ! Assign default values to an MQPMO structure
```

Mit ähnlichem Code können weitere IBM WebSphere MQ-Strukturen deklariert und initialisiert werden.

pTAL und die CRE

pTAL-Programme können die gemeinsame Laufzeitumgebung (Common Runtime Environment, CRE) nicht initialisieren und müssen daher mit einer C- oder COBOL-Hauptroutine verwendet werden.

In den mit IBM WebSphere MQ bereitgestellten pTAL-Beispielen wird eine C-Hauptroutine mit der Bezeichnung AMQSPTM0.C verwendet.

Parameter mit dem Datentyp MQCHAR

Zu den Prozeduren MQGET, MQPUT und MQPUT1 gibt es jeweils einen Parameter **Buffer** mit dem Datentyp MQCHAR .EXT. Mit diesem Parameter werden die Nachrichtendaten einer Anwendung gesendet und empfangen.

Parameter dieser Art werden in den pTAL-Beispielen als Bereiche von Zeichenfolgen angezeigt. 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 Prozedurparameter wird als `MQCHAR .EXT` deklariert, die Adresse von Daten kann jedoch als Parameter im Prozeduraufruf angegeben werden.

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.

Codierung in Visual Basic

Beachten Sie bei der Codierung von WebSphere MQ-Programmen in der Programmiersprache Visual Basic die Informationen im folgenden Abschnitt.

Anmerkung: Außerhalb der .NET-Umgebung wurde die Unterstützung von Visual Basic (VB) in WebSphere MQ auf Versionsebene V6.0 stabilisiert. Die meisten neuen Funktionen von WebSphere MQ 7.0 oder höher stehen für VB-Anwendungen nicht zur Verfügung. Wenn Sie in VB.NET programmieren, verwenden Sie die WebSphere MQ .NET-Klassen. Weitere Informationen finden Sie im Abschnitt Verwendung von .NET.

Visual Basic wird nur unter Windows unterstützt.

Um eine unbeabsichtigte Übersetzung von Binärdaten zwischen Visual Basic und WebSphere MQ zu vermeiden, verwenden Sie anstelle von MQSTRING eine MQBYTE-Definition. CMQB.BAS definiert mehrere neue MQBYTE-Typen, die einer C-Byte-Definition entsprechen, und verwendet diese in WebSphere 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 WebSphere MQ-Datenstrukturen verwiesen. Deklarieren Sie eine zusammengesetzte Struktur, die aus den beiden Komponentenstrukturen besteht, und geben Sie diese beim Aufruf an. WebSphere MQ-Unterstützung von 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.

Das IBM WebSphere MQ-Objektmodell

Das IBM WebSphere MQ-Objektmodell besteht aus Klassen, Methoden und Eigenschaften. In diesem Abschnitt erhalten Sie Informationen zu den einzelnen Konzepten.

Das IBM WebSphere MQ-Objektmodell besteht aus folgenden Komponenten:

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

Wenn Sie eine WebSphere MQ-Anwendung mit dem WebSphere MQ-Objektmodell erstellen, erstellen Sie Instanzen dieser Klassen im Programm. 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).

In den nachstehenden Abschnitten werden die einzelnen WebSphere MQ-Objektmodelle ausführlich erläutert:

- „Klassen” auf Seite 92
- „Objektverweise” auf Seite 93
- „Rückgabecodes” auf Seite 93

Klassen

Das WebSphere 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 verwendet). Es enthält eine Liste mit MQDistributionListItem-Objekten.

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

Objektverweise

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

Diese Kennungen müssen bei nachfolgenden WebSphere MQ-Aufrufen als Parameter übergeben werden. Das WebSphere 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.

Soll ich IBM WebSphere MQ Classes for Java oder IBM WebSphere MQ Classes for JMS verwenden?

Eine Java-Anwendung kann entweder IBM WebSphere MQ Classes for Java oder IBM WebSphere MQ Classes for JMS verwenden, um auf IBM WebSphere MQ -Ressourcen zuzugreifen. Jede Methode hat ihre Vorteile.

IBM WebSphere MQ Classes for Java kapselt Message Queue Interface (MQI), die native IBM WebSphere MQ -API, und verwendet dasselbe Objektmodell wie andere objektorientierte Schnittstellen, während IBM WebSphere MQ Classes for Java Message Service die JMS-Schnittstellen von Sun implementiert.

Wenn Sie mit IBM WebSphere MQ in anderen Umgebungen als Java vertraut sind und prozedurale oder objektorientierte Sprachen verwenden, können Sie Ihr vorhandenes Wissen mithilfe von IBM WebSphere MQ -Klassen für Java in die Java-Umgebung übertragen. Sie können außerdem den vollständigen Funktionsbereich von IBM WebSphere MQ nutzen, während in IBM WebSphere MQ-Klassen für JMS nicht alle Funktionen verfügbar sind.

Wenn Sie mit IBM WebSphere MQ nicht vertraut sind oder bereits über Erfahrungen mit JMS verfügen, ist es für sie möglicherweise einfacher, für den Zugriff auf IBM WebSphere MQ-Ressourcen die vertraute JMS-API zu verwenden, indem Sie IBM WebSphere MQ-Klassen für JMS verwenden. JMS ist auch ein integraler Bestandteil der Plattform Java Platform, Enterprise Edition (Java EE). Java EE -Anwendungen können Message-driven Beans (MDBs) verwenden, um Nachrichten asynchron zu verarbeiten, und MDBs können nur JMS-Nachrichten verarbeiten. JMS ist auch der Standardmechanismus für Java EE für die Interaktion mit asynchronen Messaging-Systemen wie IBM WebSphere MQ. Jeder Java EE -konforme Anwendungsserver muss einen JMS-Provider enthalten. Deshalb können Sie JMS für die Kommunikation zwischen verschiedenen Anwendungsservern verwenden oder eine Anwendung von einem JMS-Provider auf einen anderen portieren, ohne Änderungen an der Anwendung vornehmen zu müssen.

IBM WebSphere MQ-Anwendungen entwerfen

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

Beim Entwerfen einer IBM WebSphere 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 WebSphere MQ schreiben. Weitere Informationen finden Sie unter Einführung in die WebSphere MQ Administration Interface (MQAI) und unter Verwaltungsaufgaben automatisieren.

Programmiersprache
IBM WebSphere MQ unterstützt eine Reihe von prozeduralen und objektorientierten Programmiersprachen für das Schreiben von Anwendungen. Weitere Informationen hierzu finden Sie unter „Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82.

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 C verwenden, codieren Sie im 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.

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. Weitere Informationen finden Sie unter Warteschlangenmanager-Cluster.

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 „Nachrichten entwerfen” auf Seite 96.

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 WebSphere MQ-Nachricht an ein Standardziel, das von IBM WebSphere MQ publish?subscribe verwaltet wird. IBM WebSphere 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 Introduction to IBM WebSphere MQ 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.

Steuerung der IBM WebSphere MQ-Programme

Möglicherweise möchten Sie einige Programme automatisch starten oder Programme warten lassen, bis eine bestimmte Nachricht in einer Warteschlange eintrifft (mithilfe der IBM WebSphere MQ *Triggering* -Funktion, siehe „IBM WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350). Alternativ können Sie eine weitere Instanz einer Anwendung starten, wenn die Nachrichten in einer Warteschlange nicht schnell genug verarbeitet werden (mit der Funktion IBM WebSphere MQ *Instrumentierungsereignisse* , wie unter Instrumentierungsereignissebeschrieben).

Anwendung auf einem IBM WebSphere MQ-Client ausführen

In der Clientumgebung wird die vollständige MQI unterstützt, sodass nahezu jede IBM WebSphere MQ-Anwendung zur Ausführung auf einem IBM WebSphere MQ-MQI-Client erneut verbunden werden kann. Verbinden Sie die Anwendung auf dem IBM WebSphere MQ-MQI-Client mit der MQIC-Bibliothek anstatt mit der MQI-Bibliothek.

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

Weitere Informationen finden Sie unter „Anwendungen in der IBM WebSphere MQ-MQI-Clientumgebung ausführen” auf Seite 382.

Anwendungsleistung

Entwurfsentscheidungen können sich auf die Anwendungsleistung auswirken. Vorschläge zur Leistungsoptimierung von IBM WebSphere MQ-Anwendungen finden Sie unter „Anwendungsdesign und -leistung” auf Seite 97 .

Erweiterte IBM WebSphere MQ-Verfahren

Für anspruchsvollere Anwendungen empfiehlt sich unter Umständen die Nutzung erweiterter IBM WebSphere MQ-Verfahren, z. B. das Korrelieren von Antworten sowie das Generieren und Senden von IBM WebSphere MQ-Kontextinformationen. Weitere Informationen finden Sie unter „Erweiterte IBM WebSphere MQ-Verfahren” auf Seite 99.

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 WebSphere 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 343). Mit der *Persistenz*-Funktion von IBM WebSphere MQ-Nachrichten stellen Sie die Zustellung wichtiger Nachrichten sicher.

IBM WebSphere MQ-Anwendungen testen

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

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

Zugehörige Konzepte

IBM WebSphere MQ - Technischer Überblick

„Konzepte für die Anwendungsentwicklung” auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Anwendung zur Warteschlangensteuerung schreiben” auf Seite 206
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.

„Clientanwendungen schreiben” auf Seite 374
Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„.NET verwenden” auf Seite 594
WebSphere MQ-Klassen für .NET ermöglichen es einem Programm, das im .NET-Programmierungsframework geschrieben wurde, eine Verbindung zu WebSphere MQ als WebSphere MQ-MQI-Client herzustellen oder sich direkt mit einem WebSphere MQ-Server zu verbinden.

„C++ verwenden” auf Seite 668
WebSphere MQ bietet C++-Klassen, die WebSphere MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die den Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

„WebSphere MQ-Klassen für JMS verwenden” auf Seite 759
WebSphere MQ Classes for Java Message Service (WebSphere MQ Classes for JMS) ist der JMS-Anbieter, der mit WebSphere MQbereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die WebSphere MQ-Klassen für JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

„WebSphere MQ Classes for Java verwenden” auf Seite 693
Mit WebSphere MQ Classes for Java können Sie WebSphere MQ in einer Java-Umgebung verwenden. Eine Java-Anwendung kann entweder WebSphere MQ Classes for Java oder WebSphere MQ Classes for JMS verwenden, um auf WebSphere MQ -Ressourcen zuzugreifen.

„Component Object Model-Schnittstelle verwenden (WebSphere MQ Automation Classes for ActiveX)” auf Seite 1094
WebSphere MQ Automation Classes for ActiveX (MQAX) sind ActiveX-Komponenten, die Klassen bereitstellen, die Sie in Ihrer Anwendung für den Zugriff auf WebSphere MQ verwenden können.

Nachrichten entwerfen

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 WebSphere 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 WebSphere MQ for Windows oder WebSphere 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.

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
Wenn eine Nachricht unter Umständen in mehrere Warteschlangen gestellt werden muss, können Sie eine Verteilerliste verwenden oder ein Thema veröffentlichen.

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 WebSphere 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.

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 hat Auswirkungen hinsichtlich des Speichers und auch in der Form, dass 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

Unter WebSphere MQ for Windows benötigt eine Anwendung unter Umständen eine große Zahl an 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.

Erweiterte IBM WebSphere MQ-Verfahren

Bei einer einfachen IBM WebSphere MQ-Anwendung müssen Sie festlegen, welche WebSphere MQ-Objekte in dieser Anwendung verwendet werden sollen und welche Nachrichtentypen Sie verwenden möchten. 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 284).
- Durch Erstellen eines Callback-Exits, der bei Eingang einer Nachricht ausgelöst wird; siehe „Asynchrone Verarbeitung von IBM WebSphere MQ-Nachrichten” auf Seite 35.
- 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 WebSphere 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 40.

Automatisches Starten von WebSphere MQ-Programmen

Verwenden Sie WebSphere MQ *Triggering* , um ein Programm automatisch zu starten, wenn Nachrichten in einer Warteschlange ankommen.

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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350. Triggering ist jedoch nur eine Möglichkeit, ein Programm automatisch zu starten. Sie können beispielsweise ein Programm automatisch mittels Nicht-WebSphere MQ-Funktionen in einem Zeitgeber starten.

WebSphere MQ auf allen Plattformen Serviceobjekte definieren, sodass beim Start des Warteschlangenmanagers WebSphere MQ-Programme gestartet werden (siehe Service objects).

Generieren von WebSphere MQ-Berichten

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 unter „Berichtsnachrichten" auf Seite 12 beschrieben.

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 finden Sie unter Handhabung von Nachrichtenaffinitäten .

WebSphere MQ-Beispielprogramme

In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

- „Beispielprogramme für verteilte Plattformen" auf Seite 101

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung" auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Entscheiden, welche Programmiersprache verwendet werden soll" auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„IBM WebSphere MQ-Anwendungen entwerfen" auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„Anwendung zur Warteschlangensteuerung schreiben" auf Seite 206

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.

„Clientanwendungen schreiben” auf Seite 374
Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„Web-Services in WebSphere MQ verwenden” auf Seite 1004
Sie können IBM WebSphere MQ-Anwendungen für Web-Services mithilfe des IBM WebSphere MQ-Transports für SOAP oder der IBM WebSphere MQ-Bridge für HTTP entwickeln.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 295
Beginnen Sie nun, Ihre eigenen WebSphere MQ-Publish/Subscribe-Anwendungen zu entwickeln.

„IBM WebSphere MQ-Anwendung erstellen” auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

„Programmfehler behandeln” auf Seite 581
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.

Beispielprogramme für verteilte Plattformen

In diesem Thema werden die mit IBM WebSphere MQbereitgestellten Beispielprogramme beschrieben, die in C und COBOL geschrieben sind. Die Beispiele veranschaulichen typische Verwendungen des Message Queue Interface (MQI).

Die Beispielprogramme sind nicht zur Darstellung allgemeiner Programmiertechniken gedacht. Daher sind keine Fehlerprüfungen für Produktionsprogramme enthalten. Die Beispiele eignen sich aber als Basis für Ihre eigenen Message-Queuing-Programme.

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.

Beispielprogramme in C++: Im Abschnitt Verwendung von C++ finden Sie eine Beschreibung der in C++ verfügbaren Beispielprogramme.

Die Namen der Beispielprogramme beginnen mit dem Präfix `amq`. Das vierte Zeichen gibt die Programmiersprache und den Compiler an, wenn erforderlich.

s	Programmiersprache C
0	COBOL-Programmiersprache bei IBM- und Micro Focus-Compilern
i	COBOL-Programmiersprache nur bei IBM-Compilern
m	COBOL-Sprache nur bei 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. Details zum Einrichten des Warteschlangenmanagers für die Annahme von Clientverbindungen finden Sie im Abschnitt „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

Weitere Informationen zu den Beispielprogrammen finden Sie unter den folgenden Links:

- „In den Beispielprogrammen veranschaulichte Funktionen” auf Seite 102
- „Publish/Subscribe-Beispielprogramme” auf Seite 142
- „Die Put-Beispielprogramme” auf Seite 148
- „Das Distribution List-Beispielprogramm” auf Seite 134
- „Die Browse-Beispielprogramme” auf Seite 122
- „Das Beispielprogramm 'Browser'” auf Seite 123

- „Die Get-Beispielprogramme” auf Seite 136
- „Die Reference Message-Beispielprogramme” auf Seite 149
- „Die Request-Beispielprogramme” auf Seite 155
- „Die Inquire-Beispielprogramme” auf Seite 141
- „Beispielprogramm zum Abfragen der Eigenschaften eines Nachrichtenhandles” auf Seite 142
- „Die Set--Beispielprogramme” auf Seite 160
- „Die Echo-Beispielprogramme” auf Seite 135
- „Das Data-Conversion-Beispielprogramm” auf Seite 126
- „Auslösebeispielprogramme” auf Seite 164
- „Das Asynchronous Put-Beispielprogramm” auf Seite 121
- „Beispiele zur Datenbankkoordination” auf Seite 126
- „CICS-Transaktionsbeispiel” auf Seite 124
- „TUXEDO; Beispielprogramme” auf Seite 165
- „Beispielprogramm für eine Steuerroutine der Warteschlange für nicht zustellbare Nachrichten” auf Seite 134
- „Das Connect-Beispielprogramm” auf Seite 124
- „API Exit-Beispielprogramm” auf Seite 119
- „Verwenden des SSPI-Sicherheitsexits auf Windows-Systemen” auf Seite 179
- „Beispiele unter Verwendung ferner Warteschlangen ausführen” auf Seite 180
- „Das Beispielprogramm 'Clusterwarteschlangenüberwachung' (AMQSCLM)” auf Seite 180
- „Das Beispielprogramm für die Suche nach Verbindungsendpunkten (CEPL)” auf Seite 191

In den Beispielprogrammen veranschaulichte Funktionen

Dieser Abschnitt enthält eine Reihe von Tabellen, in denen die Funktionen der WebSphere MQ-Beispielprogramme aufgeführt sind.

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.

Beispiele für UNIX and Linux-Systeme

Dieser Abschnitt zeigt die Verfahren, die von den Beispielprogrammen für WebSphere MQ auf UNIX and Linux-Systemen veranschaulicht werden.

Informationen zur Speicherposition der Beispielprogramme für WebSphere MQ auf UNIX-und Linux -Systemen finden Sie unter „Vorbereiten und Ausführen von Beispielprogrammen auf UNIX-Systemen” auf Seite 117 .

Tabelle 14 auf Seite 102 Die Tabelle führt auf, welche C- und COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 14. Beispielprogramme für WebSphere MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL)

Verfahren	**C (Quelle) („1” auf Seite 105)**	**COBOL (Quelle) („2” auf Seite 105)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 105)**
Verwendung der Publish/Subscribe-Schnittstelle	amqspuba amqssuba amqssbxa	kein Beispiel	amqspub amqssub amqssbx	kein Beispiel

Tabelle 14. Beispielprogramme für WebSphere MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 105)**	**COBOL (Quelle) („2” auf Seite 105)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 105)**
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 105)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Beantworten einer Anforderungsnachricht	amqsinqa	amqminqx amqiinqx	amqsinq	kein Beispiel
Abrufen von Nachrichten (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
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha	kein Beispiel	amqsech	kein Beispiel
Referenznachrichten in eine Warteschlange einreihen („4” auf Seite 105)	amqsprma	kein Beispiel	amqsprm	amqsprmc
Referenznachrichten aus einer Warteschlange abrufen („4” auf Seite 105)	amqsgrma	kein Beispiel	amqsgrm	amqsgrmc
Referenznachricht-Kanalexit („4” auf Seite 105)	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

Tabelle 14. Beispielprogramme für WebSphere MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	**C (Quelle) („1” auf Seite 105)**	**COBOL (Quelle) („2” auf Seite 105)**	**Server (ausführbar in C)**	**Client (ausführbar in C) („3” auf Seite 105)**
Verwenden eines aufgelösten Warteschlangennamens	amqsgbr0	amq0gbr0	amqsgbr	kein Beispiel
Auslösen eines Prozesses	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Verwendung der Datenkonvertierung	(„5” auf Seite 105)	kein Beispiel	kein Beispiel	kein Beispiel
WebSphere MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL eine einzelne Datenbank auf	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	kein Beispiel	kein Beispiel
WebSphere 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 105)	amqscic0.ccs	kein Beispiel	amqscic0	kein Beispiel
Encina -Transaktion („4” auf Seite 105)	amqsxae0	kein Beispiel	amqsxae0	kein Beispiel
TUXEDO-Transaktion zum Einreihen von Nachrichten („7” auf Seite 105)	amqstxpx	kein Beispiel	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Abrufen von Nachrichten („7” auf Seite 105)	amqstxgx	kein Beispiel	kein Beispiel	kein Beispiel
Server für TUXEDO („7” auf Seite 105)	amqstxsx	kein Beispiel	kein Beispiel	kein Beispiel
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	Verzeichnis `./tools/c/Samples/dlq` („8” auf Seite 105)	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

Tabelle 14. Beispielprogramme für WebSphere MQ auf UNIX and Linux zur Veranschaulichung der Verwendung der MQI (C und COBOL) (Forts.)

Verfahren	C (Quelle) („1” auf Seite 105)	COBOL (Quelle) („2” auf Seite 105)	Server (ausführbar in C)	Client (ausführbar in C) („3” auf Seite 105)
Verwenden von Nachrichtenkonsumenten zum asynchronen Konsumieren von Nachrichten aus mehreren Warteschlangen	amqscbf0	kein Beispiel	amqscbf	amqscbfc
Angabe von SSL/TLS-Verbindungsinformationen in MQCONNX	amqssslc	kein Beispiel	Nicht zutreffend	amqssslc

Anmerkungen:

1. Die ausführbare Version der WebSphere MQ-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 WebSphere MQ-Client-Beispielprogramme stehen unter WebSphere MQ for HP-UX nicht zur Verfügung.
4. Nur für WebSphere MQ for AIX, WebSphere MQ for HP-UX und WebSphere MQ for Solaris unterstützt.
5. Auf WebSphere MQ for AIX, WebSphere MQ for HP-UX und WebSphere MQ for Solaris ist der Name dieses Programms `amqsvfc0.c`
6. CICS wird nur von WebSphere MQ for AIX und WebSphere MQ for HP-UX unterstützt.
7. TUXEDO wird von WebSphere MQ für Linux auf System p nicht unterstützt.
8. Die Quelle für die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten besteht aus mehreren Dateien und wird in einem separaten Verzeichnis bereitgestellt.

Ausführliche Informationen zur Unterstützung von UNIX and Linux-Systemen finden Sie auf der Seite mit den WebSphere MQ-Systemanforderungen unter Systemvoraussetzungen für IBM WebSphere MQ.

Beispiele für den IBM WebSphere MQ-Client für HP Integrity NonStop Server

In diesem Abschnitt werden die Verfahren gezeigt, die in den Beispielprogrammen für den IBM WebSphere MQ-Client in HP Integrity NonStop Server-Systemen veranschaulicht werden.

Tabelle 15 auf Seite 105In der Tabelle sind die bereitgestellten Quellenbeispielprogramme für C, COBOL und pTAL aufgeführt.

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL

Verfahren	C				COBOL		pTAL	
	OSS (Quelle)	OSS (Ausführbare Datei)	Guardian (Quelle)	Guardian (Ausführbare Datei)	OSS (Quelle)	Guardian (Quelle)	OSS (Quelle)	Guardian (Quelle)

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	**C**				**COBOL**		**pTAL**	
Verwendung der Publish/Subscribe-Schnittstelle	amqspuba.c amqssbxa.c amqssuba.c amqspse0.c	amqspubc amqssbxc amqssubc	MQSPUBC MQSSBXC MQSSUBC	AMQSPUBC AMQSSBXC AMQSSUBC	amq0pub0.cbl amq0sub0.cbl	MQSPUBL MQSSUBL	amqtpub0.tal amqtsub0.tal	MQSPUBT MQSSUBT
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	amqsput0.c	amqsputc	MQSPUTC	AMQSPUTC	amq0put0.cbl	MQSPUTL	amqtput0.tal	MQSPUTT
Einreihen einer einzelnen Nachricht mithilfe des MQPUT1-Aufrufs	amqsecha.c	amqsechc	MQSECHC	AMQSECHC			amqtech0.tal	MQSECHT
Nachrichten in eine Verteilerliste einreihen	amqsptl0.c	amqsptlc	MQSPTLC	AMQSPTLC	amq0ptl0.cbl	MQSPTLL		
Beantworten einer Anforderungsnachricht	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Abrufen von Nachrichten (ohne Wartezeit)	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	**C**				**COBOL**		**pTAL**	
Abrufen von Nachrichten (Wartezeit mit Zeitlimit)	amqsget0.c	amqsgetc	MQSGETC	AMQSGETC	amq0get0.cbl	MQSGETL	amqtget0.tal	MQSGETT
Abruf von Nachrichten (unbegrenzte Wartezeit)	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha.c	amqsechc	MQSECHC	AMQSECHC				
Einreihen von Referenznachrichten in eine Warteschlange	amqsprma.c	amqsprmc	MQSPRMC	AMQSPRMC				
Abruf von Referenznachrichten aus einer Warteschlange	amqsgrma.c	amqsgrmc	MQSGRMC	AMQSGRMC				
Kanalexit für Referenznachrichten	amqsqrma.c amqsxrma.c		MQSQRMC MQSXRMC					

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	C				COBOL		pTAL	
Anzeige der ersten 20 Zeichen einer Nachricht	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Anzeige vollständiger Nachrichten	amqsbcg0.c	amqsbcgc	MQSBCGC	AMQSBCGC				
Verwenden einer gemeinsam genutzten Eingabewarteschlange	amqsinqa.c	amqsinqc	MQSINQC	MQSINQC				
Verwenden einer ausschließlichen Eingabewarteschlange	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
Verwenden des MQINQ-Aufrufs	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Verwenden des MQSET-Aufrufs	amqsseta.c	amqssetc	MQSSETC	AMQSSETC				
Verwenden einer Empfangswarteschlange für Antworten	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	**C**				**COBOL**		**pTAL**	
Anforderung von Nachrichtenausnahmen	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
Akzeptieren einer abgeschnittenen Nachricht	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Verwenden eines aufgelösten Warteschlangennamens	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Auslösen eines Prozesses	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
Verwendung der Datenkonvertierung	amqsvfc0.c							
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten (1)	Verzeichnis ./samp/dlq							

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	**C**				**COBOL**		**pTAL**	
Verbindung mit einem Warteschlangenmanager über MQCONNX	amqscnxc.c	amqscnxc	MQSCNXC					
Verwendung von API-Exits	amqsaxe0.c amqsaem0.c							
Exit für Clusterlastausgleich	amqswlm0.c		MQSWLMC					
Clusterwarteschlangenüberwachung	amqsclma.c							
Asynchrones Einreihen von Nachrichten und Abrufen des Status mit dem MQSTAT-Aufruf	amqsapt0.c	amqsaptc	MQSAPTC	MQSAPTC				
Wiederverbindungsfähige Clients	amqsghac.c amqsmhac.c amqsphac.c	amqsghac amqsmhac amqsphac	MQSGHAC MQSMHAC MQSPHAC MQSFHAC	AMQSGHAC AMQSMHAC AMQSPHAC AMQSFHAC				

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	**C**				**COBOL**		**pTAL**	
Verwendung der Nachrichtenkonsumenten zum asynchronen Konsumieren von Nachrichten aus mehreren Warteschlangen	amqscbf0.c	amqscbfc						
Angabe von SSL/TLS-Verbindungsinformationen in MQCONNX	amqssslc.c	amqssslc	MQSSSLC	AMQSSSLC				
Aktivitätstrace	amq-sact0.c	amqsactc	MQSACTC	AMQ-SACTC				
Nachrichteneigenschaften	amqsiqma.c amqsstma.c	amq-siqmc amqsstmc	MQSIQMC MQSSTMC	AMQ-SIQMC AMQSSTMC				
Befehlsserver	amqss-top.c		MQSSTOC					
Protokollereignisse	amqs-log0.c	amqslogc	MQSLOGC	AMQS-LOGC				
Buchhaltung	amqs-mon0.c	amqs-monc	MQSMONC	AMQS-MONC				
Verwaltungsschnittstelle	amq-saicq.c amq-saiem.c amq-sailq.c							

Tabelle 15. Beispielprogramme für IBM WebSphere MQ auf HP Integrity NonStop Server zur Veranschaulichung der Verwendung von C, COBOL und pTAL (Forts.)

Verfahren	C				COBOL		pTAL	
Beispiel einer Hauptfunktion in der Programmiersprache C zum Aufrufen von pTAL			MQSPTMC					

Anmerkungen:

1. Die Quelle für die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten besteht aus mehreren Dateien und wird in einem separaten Verzeichnis bereitgestellt.
2. Informationen zur Entwicklung von Anwendungen für den IBM WebSphere MQ-Client auf der Plattform HP Integrity NonStop Server finden Sie in folgenden Abschnitten:
   - „Anwendung unter HP Integrity NonStop Server erstellen” auf Seite 462
     - „C-Programme in HP Integrity NonStop Server vorbereiten” auf Seite 465
     - „COBOL-Programme vorbereiten” auf Seite 466
     - „pTAL-Programme vorbereiten” auf Seite 467

Beispiel für IBM WebSphere MQ für Windows

In diesem Beispiel werden die Verwendungsmöglichkeiten der Beispielprogramme für IBM WebSphere MQ für Windows aufgezeigt.

Tabelle 16 auf Seite 112 Die Tabelle führt auf, welche C- und COBOL-Quellendateien bereitgestellt werden und ob eine ausführbare Server- oder Clientdatei enthalten ist.

Tabelle 16. Beispielprogramme für IBM WebSphere MQ für Windows, die die Verwendung der MQI (C und COBOL) veranschaulichen

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

Tabelle 16. Beispielprogramme für IBM WebSphere MQ für Windows, die die Verwendung der MQI (C und COBOL) veranschaulichen (Forts.)

Verfahren	C (Quelle)	COBOL (Quelle)	Server (ausführbar in C)	Client (ausführbar in C)
Abrufen von Nachrichten (ohne Wartezeit)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Abrufen von Nachrichten (Wartezeit mit Zeitlimit)	amqsget0	amq0get0	amqsget	amqsgetc
Abruf von Nachrichten (unbegrenzte Wartezeit)	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Abrufen von Nachrichten (mit Datenkonvertierung)	amqsecha	kein Beispiel	amqsech	amqsechc
Einreihen von Referenznachrichten in eine Warteschlange	amqsprma	kein Beispiel	amqsprm	amqsprmc
Abruf von Referenznachrichten aus einer Warteschlange	amqsgrma	kein Beispiel	amqsgrm	amqsgrmc
Kanalexit für Referenznachrichten	amqsqrma amqsxrma	kein Beispiel	amqsxrm	kein Beispiel
Anzeige der ersten 20 Zeichen einer Nachricht	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Anzeige vollständiger Nachrichten	amqsbcg0	kein Beispiel	amqsbcg	amqsbcgc
Verwenden einer gemeinsam genutzten Eingabewarteschlange	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Verwenden einer ausschließlichen Eingabewarteschlange	amqstrg0	amq0req0	amqstrg	amqstrgc
Verwenden des MQINQ-Aufrufs	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Verwenden des MQSET-Aufrufs	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
MQINQMP-Aufruf verwenden	amqsiqma	kein Beispiel	kein Beispiel	kein Beispiel
Verwenden einer Empfangswarteschlange für Antworten	amqsreq0	amq0req0	amqsreq	amqsreqc
Anforderung von Nachrichtenausnahmen	amqsreq0	amq0req0	amqsreq	amqsreqc
Akzeptieren einer abgeschnittenen Nachricht	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Verwenden eines aufgelösten Warteschlangennamens	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Auslösen eines Prozesses	amqstrg0	kein Beispiel	amqstrg	amqstrgc
Verwendung der Datenkonvertierung	amqsvfc0	kein Beispiel	kein Beispiel	kein Beispiel
WebSphere MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL eine einzelne Datenbank auf	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	kein Beispiel	kein Beispiel

Tabelle 16. Beispielprogramme für IBM WebSphere MQ für Windows, die die Verwendung der MQI (C und COBOL) veranschaulichen (Forts.)

Verfahren	C (Quelle)	COBOL (Quelle)	Server (ausführbar in C)	Client (ausführbar in C)
WebSphere MQ (koordinierende XA-konforme Datenbankmanager) ruft mittels SQL zwei Datenbanken auf	amqsxag0.c amqsxab0.sqc DB2 amqsxaf0.sqc DB2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Einreihen von Nachrichten	amqstxpx	kein Beispiel	kein Beispiel	kein Beispiel
TUXEDO-Transaktion zum Abrufen von Nachrichten	amqstxgx	kein Beispiel	kein Beispiel	kein Beispiel
Server für TUXEDO	amqstxsx	kein Beispiel	kein Beispiel	kein Beispiel
Steuerroutine der Warteschlange für nicht zustellbare Nachrichten	Verzeichnis `./tools/c/Samples/dlq` („1" auf Seite 114)	kein Beispiel	amqsdlq	kein Beispiel
In einem WebSphere MQ-Client Einreihen einer Nachricht	kein Beispiel	kein Beispiel	kein Beispiel	amqsputc
In einem WebSphere MQ-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
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
Angabe von SSL/TLS-Verbindungsinformationen in MQCONNX	amqssslc	kein Beispiel	Nicht zutreffend	amqssslc

Anmerkungen:

1. Die Quelle für die Steuerroutine der Warteschlange für nicht zustellbare Nachrichten besteht aus mehreren Dateien und wird in einem separaten Verzeichnis bereitgestellt.

Visual Basic-Beispiele für IBM WebSphere MQ für Windows

In diesem Abschnitt werden die Verwendungsmöglichkeiten der Visual Basic-Beispielprogramme für IBM WebSphere MQ für Windows aufgezeigt.

In Tabelle 17 auf Seite 115 sind die Verwendungsmöglichkeiten der Beispielprogramme für IBM WebSphere MQ für Windows aufgeführt.

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 WebSphere MQ-Server eingerichtet. Informationen darüber, wie Sie die Beispielprojekte zur Verwendung mit IBM WebSphere MQ-Clients ändern, finden Sie unter „Visual Basic-Programme unter Windows vorbereiten” auf Seite 491.

Tabelle 17. Beispielprogramme für IBM WebSphere MQ für Windows, die die Verwendung der MQI (Visual Basic) veranschaulichen

Verfahren	**Projektdateiname**
Einreihen von Nachrichten mithilfe des MQPUT-Aufrufs	amqsputb.vbp
Nachrichten mit dem MQGET-Aufruf abrufen	amqsgetb.vbp
Warteschlange mit dem MQGET-Aufruf durchsuchen	amqsbcgb.vbp
Einfaches MQGET- und MQPUT-Beispiel (Client)	mqtrivc.vbp
Einfaches MQGET- und MQPUT-Beispiel (Server)	mqtrivs.vbp
Zeichenfolgen und benutzerdefinierte Strukturen mit MQPUT und MQGET einreihen und abrufen	strings.vbp
Kanal mittels PCF-Strukturen starten und stoppen	pcfsamp.vbp
Warteschlange mit der MQAI erstellen	amqsaicq.vbp
Warteschlangen eines Warteschlangenmanagers mit der MQAI auflisten	amqsailq.vbp
Ereignisse mit der MQAI überwachen	amqsaiem.vbp

Beispielprogramme vorbereiten und ausführen

In diesem Abschnitt wird beschrieben, wie Sie Ihren Warteschlangenmanager so konfigurieren, dass er eingehende Verbindungsanforderungen von Anwendungen, die im Clientmodus ausgeführt werden, sicher annehmen kann.

Vorbereitende Schritte

Stellen Sie sicher, dass der Warteschlangenmanager vorhanden und gestartet ist. Ermitteln Sie wie folgt, ob Kanalauthentifizierungssätze aktiviert sind:

```
DISPLAY QMGR CHLAUTH
```

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. Falls Ihr Warteschlangenmanager keine Kanalauthentifizierungsdatensätze verwendet, kann Schritt „4” auf Seite 116 durch eine alternative Authentifizierungsmethode (beispielsweise einen Sicherheitsexit) ersetzt werden, wobei für MCAUSER die *nicht_privilegierte_Benutzer-ID* aus Schritt „1” auf Seite 116 festgelegt wird.

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:

   qmgr ist der Name des Warteschlangenmanagers.
   nnnn ist die von Ihnen ausgewählte Portnummer.

   a) Unter UNIX und Windows:

   ```
   runmqlsr -t tcp -m qmgr -p 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, die Sie in Schritt „1” auf Seite 116 erhalten haben.
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) Unter UNIX und Windows geben Sie hierzu die folgenden MQSC-Befehle aus:

   ```
   SET AUTHREC OBJTYPE(QMGR) PRINCIPAL('non-privileged-user-id') +
   AUTHADD(CONNECT, INQ)
   ```

   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID, die Sie in Schritt „1” auf Seite 116 erhalten haben.

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) Unter UNIX und Windows geben Sie hierzu die folgenden MQSC-Befehle aus:

   ```
   SET AUTHREC PROFILE('queue-name') OBJTYPE(QUEUE) +
   PRINCIPAL('non-privileged-user-id') AUTHADD(PUT, GET, INQ, BROWSE)
   ```

   Warteschlangenname ist der Name Ihrer Warteschlange.

   Nicht_privilegierte_Benutzer-ID ist die Benutzer-ID, die Sie in Schritt „1” auf Seite 116 erhalten haben.

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) Unter UNIX und Windows geben Sie hierzu die folgenden 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, die Sie in Schritt „1” auf Seite 116 erhalten haben.

   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 Tasks

Zugriff auf ein WebSphere MQ -Objekt auf UNIX-oder Linux -Systemen und Windows erteilen

Zugehörige Verweise

SET CHLAUTH

CHANNEL DEFINE CHANNEL

QLOCAL DEFINIER

SET AUTHREC

Vorbereiten und Ausführen von Beispielprogrammen auf UNIX-Systemen

Tabelle 18. Verzeichnisse der Beispiele für WebSphere 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 WebSphere MQ installiert ist.	

Die Beispieldateien für WebSphere MQ auf UNIX and Linux-Systemen finden Sie in den unter Tabelle 18 auf Seite 117 aufgeführten Verzeichnissen, wenn die Standardeinstellungen installiert wurden. Verwenden Sie zum Ausführen der Beispielprogramme entweder die bereitgestellten ausführbaren Versionen

oder kompilieren Sie die Quellenversionen wie jede andere Anwendung mithilfe eines ANSI-Compilers. Weitere Informationen hierzu erhalten Sie unter „Beispielprogramme ausführen” auf Seite 118.

Vorbereiten und Ausführen von Beispielprogrammen auf Windows-Systemen

Tabelle 19. Verzeichnisse der Beispiele für WebSphere MQ auf Windows-Systemen

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
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-Beispiele	*MQ_INSTALLATION_PATH*\Tools\dotnet\Samples

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

Anmerkung:

1. Für einige ausführbare C-Beispieldateien stehen 64-Bit-Versionen zur Verfügung.

Die Beispieldateien für WebSphere MQ for Windows befinden sich in den Verzeichnissen, die im Abschnitt Tabelle 19 auf Seite 118 aufgelistet sind, wenn bei der Installation die Standardwerte verwendet wurden. Das Installationslaufwerk nimmt standardmäßig den Wert < c: > an. Verwenden Sie zum Ausführen der Beispiele entweder die bereitgestellten ausführbaren Versionen oder kompilieren Sie die Quellenversionen wie alle anderen WebSphere MQ for Windows -Anwendungen. Informationen zur Vorgehensweise finden Sie in „Beispielprogramme ausführen” auf Seite 118.

Beispielprogramme ausführen

Ziehen Sie diesen Abschnitt zu Rate, wenn Sie die Beispielprogramme auf verschiedenen Plattformen ausführen möchten.

Bevor Sie eines der Beispielprogramme ausführen können, erstellen Sie zunächst einen Warteschlangenmanager und richten Sie die Standarddefinitionen ein. Dieses Vorgehen wird unter Verwaltung erläutert.

Auf Windows-, UNIX-und Linux -Plattformen

Die Beispielprogramme benötigen mehrere Warteschlangen, mit denen sie arbeiten. Entweder verwenden Sie eigene Warteschlangen oder Sie erstellen eine Reihe neuer Warteschlangen, indem Sie die MQSC-Beispieldatei 'amqscos0.tst' ausführen.

Geben Sie dazu auf Systemen mit UNIX and Linux Folgendes ein:

• `runmqsc QManagerName <amqscos0.tst >/tmp/sampobj.out`

Prüfen Sie die Datei `sampobj.out`, um sicherzustellen, dass keine Fehler vorliegen.

Geben Sie dazu auf 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.

Jetzt können Sie die Beispielanwendungen ausführen. Geben Sie den Namen der Beispielanwendung ein, gefolgt von beliebigen Parametern, beispielsweise:

- `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.

Lesen Sie die Beschreibung zu den erforderlichen Parametern der einzelnen Beispielprogramme.

Länge des Warteschlangennamens

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.

Inquire-, Set- und Echo-Beispiele

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

Auf Windows-, UNIX and Linux -Systemen geschieht dies durch Bearbeiten der Datei `amqscos0.tst` und Ändern der Namen der ausführbaren C-Dateien in die Namen der ausführbaren COBOL-Dateien, bevor der Befehl **runmqsc** wie oben gezeigt verwendet wird.

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 413.

Quelle
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 WebSphere 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 WebSphere 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: *Verbindungswiederholungstyp* 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 WebSphere 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 WebSphere 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 veranschaulichte Funktionen” auf Seite 102.

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 veranschaulichte Funktionen” auf Seite 102.

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.

UNIX-, Linux- und Windows-Systeme

Dieser Abschnitt enthält Informationen zu den Browse-Beispielprogrammen auf UNIX-, Linux- und Windows-Systemen.

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.

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 veranschaulichte Funktionen” auf Seite 102.

Für das Programm können die folgenden Parameter angegeben werden:

1. Den Namen der Quellenwarteschlange
2. Der Name des Warteschlangenmanagers.
3. Ein optionaler Parameter für Eigenschaften

Die ersten zwei Eingabeparameter für dieses Programm sind obligatorisch. Sie haben beispielsweise die folgenden Möglichkeiten, das Programm zu starten:

- `amqsbcg myqueue qmanagername`
- `amqsbcgc myqueue qmanagername`

Dabei steht `myqueue` für den Namen der Warteschlange, in der nach den Nachrichten gesucht wird, und `qmanagername` gibt den Warteschlangenmanager an, 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)

Folgende Werte sind für den Eigenschaftenparameter zulässig:

Wert	Verhalten
0	Standardverhalten wie in V6. Die Eigenschaften, die der Anwendung bereitgestellt werden, hängen vom Warteschlangenattribut *PropertyControl* ab, von dem die Nachricht abgerufen wird.

Wert	Verhalten

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_COMPATIBILITY ist angegeben, sodass alle Eigenschaften in den Nachrichtendaten zurückgegeben werden können, falls eine Eigenschaft der Version 6 vorhanden ist. Anderenfalls werden die Eigenschaften verworfen.

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.

Im Abschnitt Verwaltung finden Sie ein Beispiel für die Ausgabe dieses Dienstprogramms:

CICS-Transaktionsbeispiel

Bereitstellung des CICS-Transaktionsbeispielprogramms 'amqscic0.ccs' für den Quellcode und des Beispielprogramms 'amqscic0' für die ausführbare Version. Sie können Transaktionen mithilfe der Standardfunktionen von CICS erstellen.

Ausführliche Informationen zu den Befehlen für Ihre jeweilige Plattform finden Sie unter „IBM WebSphere MQ-Anwendung erstellen” auf Seite 456.

Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE auf dem Standardwarteschlangenmanager und stellt sie in die lokale Warteschlange, deren Name im Übertragungsheader der Nachricht angegeben ist. 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]] [QMgrName]
```

Die Parameter sind optional und ihre Reihenfolge ist unwichtig - mit Ausnahme von 'QMgrName'. Dieser Parameter muss ganz zum Schluss angegeben werden. Es gibt folgende Parameter:

ConnName
Der TCP/IP-Verbindungsname des Server-Warteschlangenmanagers

SvrconnChannelName
Name des Serververbindungskanals

QMgrName
Name des Zielwarteschlangenmanagers

Wenn Sie den TCP/IP-Verbindungsnamen nicht angeben, wird MQCONNX mit *ClientConnPtr* auf NULL ausgegeben. Wenn Sie den TCP/IP-Verbindungsnamen, aber keinen Serververbindungskanal angeben (umgekehrt ist dies nicht zulässig), verwendet das Beispielprogramm den Namen SYSTEM.DEF.SVRCONN. 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 veranschaulichte Funktionen” auf Seite 102.

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 WebSphere MQ Aktualisierungen von WebSphere 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 WebSphere 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 WebSphere MQ-Arbeitseinheit und zeigen den Zugriff auf mehrere Datenbanken. Diese Beispiele zeigen die Verwendung des MQBEGIN-Aufrufs sowie gemischter SQL- und WebSphere MQ-Aufrufe und erläutern, wo und wann eine Datenverbindung hergestellt wird.

Abbildung 18 auf Seite 127 zeigt, wie Sie mit den bereitgestellten Beispielen Datenbanken aktualisieren:

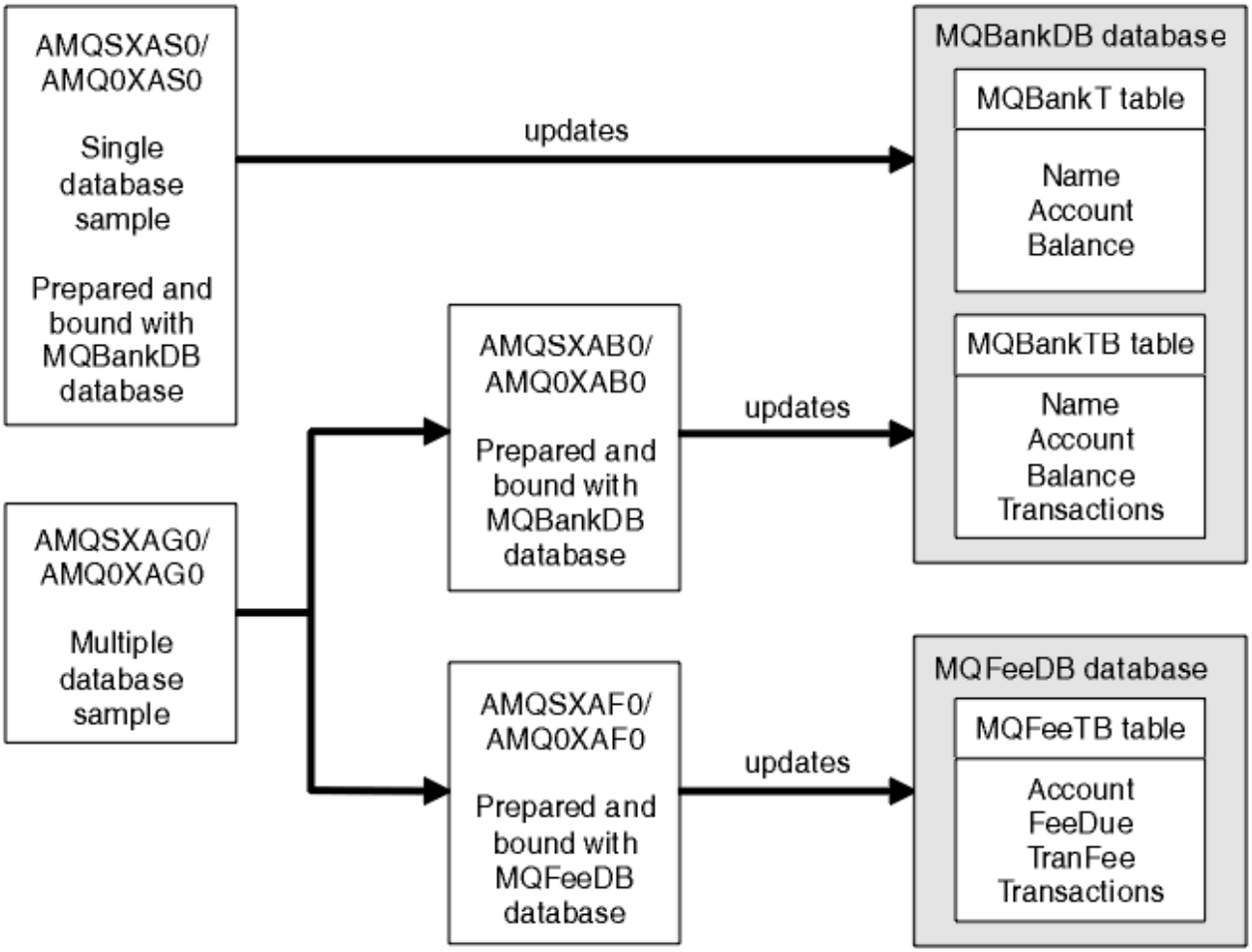


Abbildung 18. Beispiele zur Datenbankkoordination

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 WebSphere 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 WebSphere 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 WebSphere MQ enthält.

Kopieren Sie die folgenden Dateien in ein temporäres Verzeichnis:

- Die Datei `amqsxag0.c` aus der WebSphere 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`-Quelldateien `amqsxas0.sqc, amqsxaf0.sqc` und `amqsxab0.sqc` erhalten haben
- Die Dateien `util.c` und `util.h` aus der 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 -IMQ_INSTALLATION_PATH/inc -I
<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- HP-UX

```
cc -Aa +z -IMQ_INSTALLATION_PATH/inc -I
<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- Solaris

```
cc -Aa -KPIC -mt -IMQ_INSTALLATION_PATH
/inc -I<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- Windows-Systeme

```
cl /c /IMQ_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 -LMQ_INSTALLATION_PATH/lib
 -lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- HP-UX Revision 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_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 -LMQ_INSTALLATION_PATH/lib
 -lmqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows-Systeme

```
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 folgenden Kompilier- und Verknüpfungsbefehl für Ihre entsprechende Plattform:

- AIX

```
xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2
 -LMQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- HP-UX Revision 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread
 -lcl /lib/crt0.o util.o amqsxas0.o -o amqsxas0
```

- Solaris

```
cc -mt -L<DB2TOP>/lib -ldb2-LMQ_INSTALLATION_PATH/lib
 -lqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxas0.o -o amqsxas0
```

- Windows-Systeme

```
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 finden Sie im Abschnitt „Szenario 1: Der Warteschlangenmanager nimmt die Koordinierung vor" auf Seite 45.

Die folgenden Titel stellen Informationen zum Ausführen der Beispielprogramme in C und COBOL bereit:

- „C-Beispiele" auf Seite 132
- „COBOL-Beispiele" auf Seite 133

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 WebSphere MQ-Fehlerprotokolldatei ein, sondern in 'stdout'.

Im Handbuch Verwaltung 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 veranschaulichte Funktionen” auf Seite 102.

Die Programme wurden als Auslöserprogramme konzipiert.

Auf UNIX-, Linux- und Windows-Systemen ist ihre einzige Eingabe eine MQTMC2-Struktur (Auslösenachricht), die die Namen einer Zielwarteschlange und des Warteschlangenmanagers enthält. Die COBOL-Version verwendet den Standardwarteschlangenmanager.

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. (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, 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.

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 veranschaulichte Funktionen” auf Seite 102.

Gestaltung des Get-Beispielprogramms

Das Programm öffnet die Zielwarteschlange mit dem MQOPEN-Aufruf und der Option MQOO_INPUT_AS_Q_DEF. Wenn es die Warteschlange nicht öffnen kann, zeigt das Programm eine Fehlernachricht an, die den vom MQOPEN-Aufruf zurückgegebenen Ursachencode enthält.

Das Programm entfernt jede Nachricht mit dem MQGET-Aufruf aus der Warteschlange und zeigt dann die Daten an, die in der jeweiligen Nachricht enthalten sind. Der MQGET-Aufruf verwendet die MQGMO_WAIT-Option und gibt einen *WaitInterval* von 15 Sekunden an, sodass das Programm für diesen Zeitraum wartet, wenn keine Nachricht in der Warteschlange vorhanden ist. Wenn keine Nachricht eintrifft, bevor dieses Intervall abläuft, schlägt der Aufruf fehl und gibt den Ursachencode MQRC_NO_MSG_AVAILABLE zurück.

Das Programm veranschaulicht, wie die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf gelöscht werden müssen, da der Aufruf diese Felder auf die Werte setzt, die in den abgerufenen Nachrichten enthalten sind. Durch das Löschen dieser Felder rufen aufeinanderfolgende MQGET-Aufrufe die Nachrichten in der Reihenfolge ab, in der diese in der Warteschlange gehalten werden.

Der MQGET-Aufruf gibt einen Puffer mit einer festen Größe an. Wenn eine Nachricht länger als dieser Puffer ist, schlägt der Aufruf fehl und das Programm hält an.

Das Programm wird fortgesetzt, bis der MQGET-Aufruf entweder den Ursachencode MQRC_NO_MSG_AVAILABLE zurückgibt oder der MQGET-Aufruf fehlschlägt. Wenn der Aufruf fehlschlägt, zeigt das Programm eine Fehlernachricht mit dem Ursachencode an.

Das Programm schließt dann die Warteschlange mithilfe des MQCLOSE-Aufrufs.

Beispielprogramme 'amqsget' und 'amqsgetc' ausführen

Die Programme verwenden jeweils zwei 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.

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.

Wird qmanagername ausgeschlossen, setzen die Programme den Standard voraus oder - falls es sich um den MQI-Client handelt - den von einer Umgebungsvariablen oder der Clientkanaldefinitionsdatei angegebenen Warteschlangenmanager.

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 einrichten und steuern” auf Seite 139).

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 WebSphere 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 WebSphere 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 WebSphere 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 Seitenwarteschlangenname InTransaktionszähler Wiederholungszähler*** **(0|1|2)**

- **amqsfhac** ist eine IBM WebSphere MQ MQI client-Anwendung Sie prüft, ob ein IBM WebSphere MQ-Multi-Instanz-Warteschlangenmanager, der vernetzten Speicher verwendet (z. B. ein NAS- oder Clusterdateisystem), die Integrität der Daten beibehält. Führen Sie die unter Verhalten des gemeinsam genutzten Dateisystems überprüfen beschriebenen Schritte aus, um **amqsfhac** auszuführen.

- Bei der Verbindungsherstellung zu *Warteschlangenmanagername* verwendet es die Option `MQCNO_RECONNECT_Q_MGR` und stellt die Verbindung bei Ausfall des Warteschlangenmanagers automatisch wieder her.
- Es reiht die persistenten Nachrichten *InTransaktionszähler***Wiederholungszähler* in *Warteschlangenname* ein. In dieser Zeit verursachen Sie den beliebig häufigen Ausfall des Warteschlangenmanagers. Jedes Mal stellt **amqsfhac** die Verbindung zum Warteschlangenmanager wieder her und setzt den Vorgang fort. Der Test soll sicherstellen, dass keine Nachrichten verloren gehen.
- Es wird jeweils die Anzahl von *InTransaktionszähler* Nachrichten innerhalb einer Transaktion eingereiht. Die Transaktion wird *Wiederholungszähler* Male wiederholt. Schlägt eine Transaktion fehl, macht **amqsfhac** sie rückgängig und übergibt sie erneut, wenn **amqsfhac** die Verbindung zum Warteschlangenmanager wieder herstellt.
- Zudem reiht es Nachrichten in *Seitenwarteschlangenname* ein. Mittels *Seitenwarteschlangenname* prüft es, ob alle Nachrichten aus *Warteschlangenname* erfolgreich festgeschrieben oder rückgängig gemacht wurden. Wird eine Inkonsistenz festgestellt, gibt es eine entsprechende Fehlernachricht aus.
- Variieren Sie die Anzahl des Ausgabe-Tracing von **amqsfhac**, indem Sie den letzten Parameter auf `(0|1|2)` setzen.

  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. Weitere Informationen finden Sie unter Clientinstallation überprüfen.

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 19 auf Seite 138.

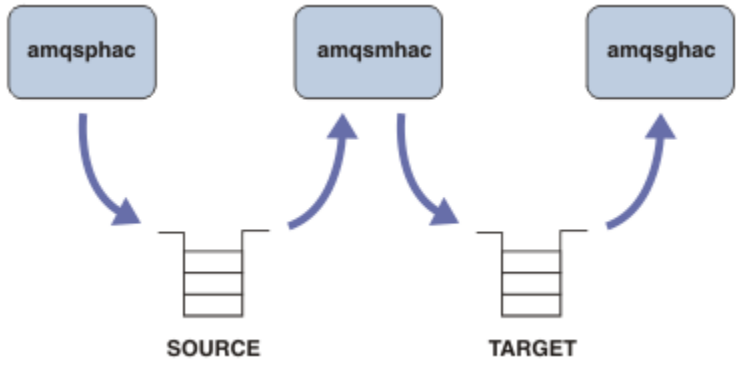


Abbildung 19. Beispiele zu wiederverbindbaren Clients

Gehen Sie zum Ausführen der Beispielprogramme wie folgt vor:

1. Erstellen Sie eine Datei namens `hasamples.tst`, die folgende Befehle enthält:

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. Geben Sie an der Eingabeaufforderung die folgenden Befehle ein:
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. Setzen Sie die Umgebungsvariable **MQCHLLIB** auf den Pfad zur AMQCLCHL.TAB-Clientkanaldefinitionsdatei, z. B. `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Öffnen Sie drei neue Fenster mit dem festgelegten Parameter **MQCHLLIB**. Geben Sie beispielsweise unter Windows dreimal 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 140 in „Warteschlangenmanager einrichten und steuern” auf Seite 139.)
5. Geben Sie den Befehl `endmqm -r -p QM1` ein, um den Warteschlangenmanager zu stoppen, und lassen Sie dann zu, dass die Clients die Verbindung wiederherstellen.
6. Geben Sie den Befehl `strmqm QM1` ein, um den Warteschlangenmanager neu zu starten.

Die Ergebnisse der Ausführung der Beispiele **amqsghac**, **amqsphac** und **amqsmhac** unter Windows werden in den folgenden Beispielen angezeigt.

Warteschlangenmanager einrichten und steuern

1. Erstellen Sie den Warteschlangenmanager.

```
C:\>crtmqm QM1
WebSphere MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Merken Sie sich das Datenverzeichnis, um die Variable **MQCHLLIB** später festzulegen.
2. Starten Sie den Warteschlangenmanager.

```
C:\>strmqm QM1
WebSphere 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'.
WebSphere MQ queue manager 'QM1' started.
```

3. Erstellen Sie die Warteschlangen und Kanäle, ändern Sie den Listener-Port und starten Sie Listener und Kanal.

```
C:\>runmqsc QM1 < hasamples.tst
5724-H72 (C) Copyright IBM Corp. 1994, 2024. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: WebSphere MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: WebSphere MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN) RE
```

```
PLACE
AMQ8014: WebSphere MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: WebSphere MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: WebSphere MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start WebSphere MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start WebSphere MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Machen Sie den Clients die Clientkanaltabelle bekannt.

   Verwenden Sie das Datenverzeichnis, das in Schritt „1” auf Seite 139 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.
WebSphere MQ queue manager 'QM1' ending.
WebSphere MQ queue manager 'QM1' ended.

C:\>strmqm QM1
WebSphere 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'.
WebSphere MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
<Message 3>
message <Message 4>
message <Message 5>
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message <Message 1>
```

```
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message <Message 3>
message <Message 4>
message <Message 5>
```

Zugehörige Tasks

Verhalten eines gemeinsam genutzten Dateisystems überprüfen

Zugehörige Verweise

amqmfsck (Dateisystemprüfung)

Die Inquire-Beispielprogramme

Die Inquire-Beispielprogramme fragen über einen MQINQ-Aufruf einige Attribute der Warteschlange ab.

Die Namen dieser Programme finden Sie unter „In den Beispielprogrammen veranschaulichte Funktionen” auf Seite 102.

Diese Programme sollen als Auslöserprogramme ausgeführt werden. Ihre einzige Eingabe ist also eine MQTMC2-Struktur (Auslösenachricht) für IBM i-, Windows- und UNIX and Linux-Systeme. 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. Die Beispielwarteschlange verfügt über den Auslösertyp FIRST; wenn also bereits Nachrichten in der Warteschlange enthalten sind, bevor Sie das Request-Beispiel ausführen, wird das Inquire-Beispiel nicht von den Nachrichten ausgelöst, die Sie senden.

Wenn Sie die Definition ordnungsgemäß festlegen müssen:

- Starten Sie auf UNIX-, Linux- und Windows-Systemen das Programm **runmqtrm** in einer Sitzung und anschließend das Programm `amqsreq` in einer anderen Sitzung.

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.

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:\Program Files\IBM\WebSphere 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 WebSphere MQ.

Drei Beispielprogramme in der Programmiersprache C veranschaulichen die Programmierung der Publish/Subscribe-Schnittstelle von WebSphere 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 WebSphere MQ in com.ibm.mq.jar und die Publish/Subscribe-Schnittstelle von JMS in com.ibm.mqjms. Die JMS-Beispiele 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 „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

Der Publisher stellt eine Verbindung zum Standardwarteschlangenmanager her und antwortet mit der Ausgabe `target topic is mytopic` . Jede Zeile, die Sie ab sofort in dieses Fenster eingeben, wird in `mytopic` veröffentlicht .

Öffnen Sie ein weiteres Befehlsfenster in demselben Verzeichnis und führen Sie das Subskribentenprogramm `amqssub` aus. Geben Sie dabei denselben Themennamen und einen optionalen Warteschlangenmanagernamen an, 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 20 auf Seite 143 sind sechs Beispielprogrammtypen aufgeführt.

Tabelle 20. Kategorien veralteter Publish/Subscribe-Beispielprogramme in Programmiersprache C

Kategorie	Programme	Kommentare
RFH1	`amqssr1a.c` `amqspr1a.c`	Einfaches Publish/Subscribe-Beispiel, erstellt mittels RFH1-Formatnachrichten.
RFH2	`amqssr2a.c` `amqspr2a.c`	Einfaches Publish/Subscribe-Beispiel, erstellt mittels RFH2-Formatnachrichten.
MQAI-Beispielprogramme	`amqsppca.c` `amqsspca.c`	Einfaches Publish/Subscribe-Beispielprogramm, erstellt mittels PCF-Befehlen und der MQAI-Befehlsschnittstelle.
MA0C-Ergebnisdienst unter Verwendung von RFH1	`amqsgama.c` `amqsresa.c`	Ergebnisdienst, erstellt mit RFH1-Headern 1. Erfordert die Definition der Warteschlangen in `amqsgama.tst` und `amqsresa.tst` 2. `amqsresa` muss vor `amqsgama` gestartet werden.
MA0C-Ergebnisdienst unter Verwendung von RFH2	`amqsgr2a.c` `amqsrr2a.c`	Ergebnisdienst, erstellt mit RFH2-Headern 1. Erfordert die Definition der Warteschlangen in `amqsgama.tst` und `amqsresa.tst` 2. `amqsresa` muss vor `amqsgama` gestartet werden.

Tabelle 20. Kategorien veralteter Publish/Subscribe-Beispielprogramme in Programmiersprache C (Forts.)		
Kategorie	**Programme**	**Kommentare**
Beispielprogramm für Publish/Subscribe in einem Routing-Exit	`amqspsra.c`	Veranschaulicht die Änderung des Ziels der Warteschlange bzw. des Warteschlangenmanagers für eine Publish/Subscribe-Nachricht in einem Routing-Exit.

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 „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

Führen Sie das Beispielprogramm über den Java-Befehl in der Befehlszeile aus, wenn Sie eine konfigurierte Java-Umgebung verwenden. Sie können das Beispielprogramm auch über den WebSphere 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 144, 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 „Beispielprogramme vorbereiten und ausführen” auf Seite 115 aus.

Informationen zu diesem Vorgang

Das Java Publish/Subscribe-Beispielprogramm ist ein WebSphere 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 allgemein, wie Sie Parameter bereitstellen und das Beispielprogramm an verschiedene WebSphere MQ-Konfigurationen anpassen. Das Beispiel wird unter Windows ausgeführt. Die Dateipfade variieren je nach verwendetet Plattform.

Vorgehensweise

1. Java-Beispielprogramme importieren
   a) Klicken Sie in der Workbench auf **Fenster** > **Perspektive öffnen** > **Andere** > **Java** und dann 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 **Neu** > **Java-Projekt**.

d) Geben Sie im Feld **Project name** `MQ Java Samplesein`. 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` , wobei *MQ_INSTALLATION_PATH* der WebSphere MQ -Installationsordner ist, 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) Auswählen **Importieren ...** > **Allgemein** > **Dateisystem** > **Weiter** > **Durchsuchen**... Navigieren Sie zum Pfad *MQ_INSTALLATION_PATH*`\tools\wmqjava\samples` , wobei *MQ_INSTALLATION_PATH* das Installationsverzeichnis von WebSphere MQ ist.

k) Klicken Sie in der Anzeige **Import** (Importieren) Abbildung 20 auf Seite 146 auf `samples` (aktivieren Sie nicht das Kontrollkästchen).

l) Wählen Sie die Datei `MQPubSubApiSample.java` aus. Das Feld **Into folder** sollte `MQ Java Samples/srcenthalten`. Klicken Sie auf **Fertigstellen**.

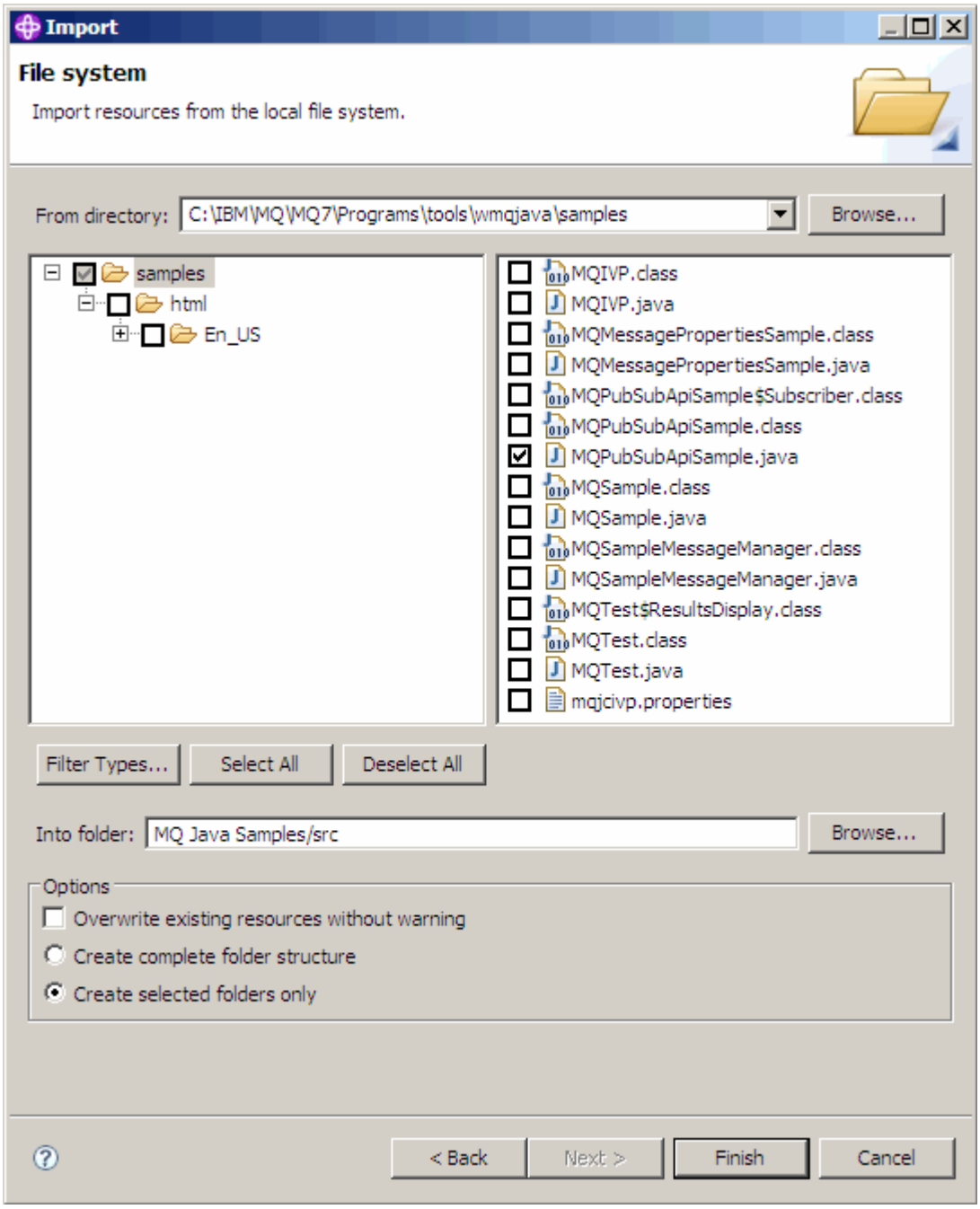


Abbildung 20. Dateisystemimport

2. Führen Sie das Publish/Subscribe-Beispielprogramm aus.

   Je nachdem, ob Sie die Standardparameter ändern müssen, kann das Programm auf zwei Arten ausgeführt werden.

   - So führen Sie das Programm aus, wenn keine Änderungen erforderlich sind:
     - Erweitern Sie im Hauptmenü des Arbeitsbereichs den Ordner `src` . Klicken Sie mit der rechten Maustaste auf **MQPubSubApiSample.java Run-as** > **1. Java-Anwendung**
   - So führen Sie das Programm mit Parametern oder mit an Ihre Umgebung angepasstem Quellcode aus:
     - Öffnen Sie die Datei `MQPubSubApiSample.java` und prüfen Sie den Konstruktor `MQPubSubApiSample`.

– Ändern Sie die Attribute des Programms.

  Diese Attribute können mit dem '-D JVM'-Switch geändert werden oder indem Sie einen Standardwert für die System-Eigenschaft angeben. Dazu muss der Quellcode bearbeitet werden.

  - topicObject
  - queueManagerName
  - subscriberCount

  Diese Attribute können nur durch Bearbeiten des Quellcodes im Konstruktor geändert werden.

  - Hostname
  - port
  - Kanal

  Zum Festlegen der Systemeigenschaften 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 **Ausführen** > **Dialog 'Ausführen' öffnen**. Doppelklicken Sie in **Anwendungen erstellen, verwalten und ausführen** auf Java-Anwendung und klicken Sie auf die Registerkarte **(x) = Argumente** .
c. Geben Sie im Teilfenster **VM arguments:** `-D` ein und fügen Sie den Namen System.property , `com.ibm.mq.pubSubSample.queueManagerName`, gefolgt von =QM3, ein. Klicken Sie auf **Anwenden** > **Ausführen**.
d. Fügen Sie weitere Argumente als eine durch Kommas getrennte Liste oder als zusätzliche Zeilen im Teilfenster (ohne Kommatrennzeichen) hinzu.

   Beispiel:`-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

Das Publish Exit-Beispielprogramm

AMQSPSE0 ist das C-Beispielprogramm eines Exits zum Abfangen einer Veröffentlichung, bevor sie einem Subskribenten bereitgestellt wird. Der Exit kann beispielsweise die Nachrichtenheader, Nutzdaten oder das Ziel ändern oder verhindern, dass die Nachricht für einen Subskribenten veröffentlicht wird.

Gehen Sie zum Ausführen des Beispielprogramms wie folgt vor:

1. Konfigurieren Sie den Warteschlangenmanager:
   - Fügen Sie auf UNIX and Linux-Systemen der Datei `qm.ini` eine Zeilengruppe wie die nachfolgende hinzu:

```
PublishSubscribe:
           PublishExitPath=<Module>
           PublishExitFunction=EntryPoint
```

   Das Modul steht unter *MQ_INSTALLATION_PATH*`/samp/bin/amqspse.`*MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist. Legen Sie unter Windows die entsprechenden Attribute in der Registry fest.
2. Stellen Sie sicher, dass WebSphere MQ auf das Modul zugreifen kann.
3. Starten Sie den Warteschlangenmanager, um die Konfiguration zu übernehmen.

4. Beschreiben Sie in dem zu verfolgenden Anwendungsprozess den Pfad, in den die Tracedateien geschrieben werden sollen. Beispiel:
   - 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
     ```

   - Stellen Sie unter Windows sicher, dass das Verzeichnis `C:\temp` vorhanden ist und legen Sie die 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 veranschaulichte Funktionen” auf Seite 102.

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

Die Programme verwenden jeweils zwei Parameter:

1. Name der Zielwarteschlange (erforderlich)
2. Name des Warteschlangenmanagers (optional)

Wenn ein Warteschlangenmanager nicht angegeben wird, stellt 'amqsput' eine Verbindung zum Standardwarteschlangenmanager her und 'amqsputc' stellt eine Verbindung zu dem Warteschlangenmanager her, der durch eine Umgebungsvariable oder in der Definitionsdatei des Clientkanals angegeben ist. 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

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.

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 WebSphere 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 unseres Beispielprogramms 'amqsxrm' einen eigenen Exit. Die Reference Message-Beispielprogramme werden in den folgenden Abschnitten erläutert.

Welche Version des Reference Message-Exit-Beispielprogramms verwendet werden kann, richtet sich nach der Plattform, auf der der Kanal aktiv ist. Verwenden Sie auf allen Plattformen 'amqsxrma' auf der Sendeseite. Verwenden Sie auf der Empfangsseite 'amqsxrma', wenn der Empfänger auf einem WebSphere MQ-Produkt ausgeführt wird (ausgenommen WebSphere MQ for IBM i ausgeführt wird.

Referenznachrichtenbeispiel ausführen

In diesem Abschnitt erfahren Sie, wie Sie die Beispielprogramme der Referenznachrichten ausführen.

Die Referenznachrichtenbeispiele werden wie folgt ausgeführt:

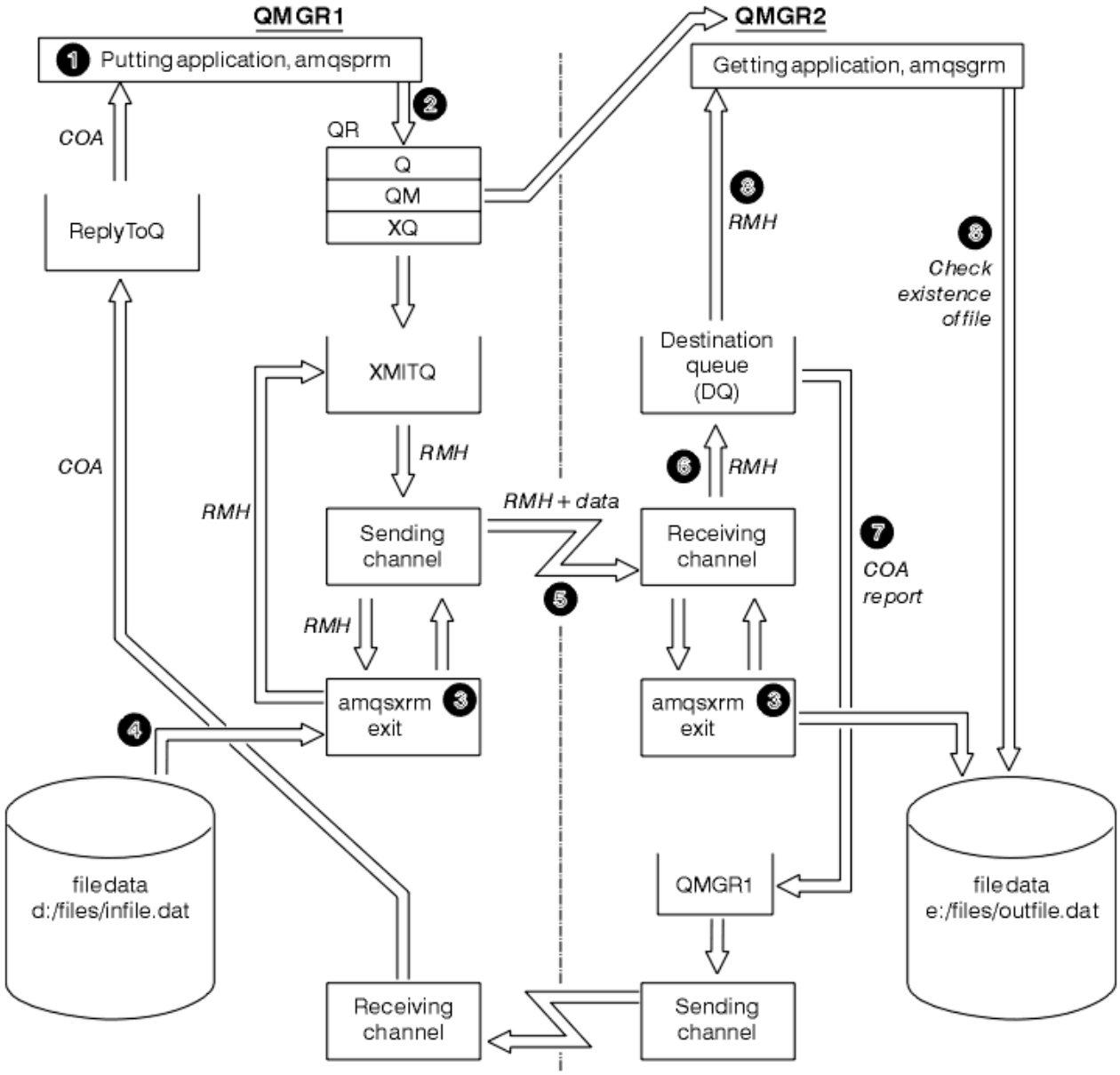


Abbildung 21. Referenznachrichtenbeispiel ausführen

1. Richten Sie die Umgebung zum Starten der Empfangsprogramme, Kanäle und Auslösemonitore ein und definieren Sie Ihre Kanäle und Warteschlangen.

   Bei der Erläuterung der Einrichtung des Referenznachrichtbeispiels werden die sendende Maschine als MACHINE1 mit einem Warteschlangenmanager namens QMGR1 und die empfangende Maschine als MACHINE2 mit dem Warteschlangenmanager 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), 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 die Verzeichnisstruktur entsprechend des Maschinentyps und des Installationspfads des WebSphere MQ-Produkts wiedergibt.

Die MQSC-Definitionen haben einen AIX -Stil für die Definition der Exits verwendet. 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
```

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)
```

2. Nachdem die WebSphere 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 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 Name der Warteschlange

-g Name des Warteschlangenmanagers mit der im Parameter '-q' definierten Warteschlange. Standardmäßig ist dies der im Parameter '-m' angegebene Warteschlangenmanager.

-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
```

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. Für WebSphere MQ for Windows ist dies wie folgt definiert:

```
msgexit('pathname\amqsxrm.dll(MsgExit)')
```

Für WebSphere MQ for AIX, WebSphere MQ for HP-UX und WebSphere MQ for Solaris ist dies wie folgt definiert:

```
msgexit('pathname/amqsxrm(MsgExit)')
```

Geben Sie immer den vollständigen Pfadnamen an. Wenn Sie ihn übergehen, wird angenommen, dass sich das Programm in dem in der Datei `qm.ini` angegebenen Pfad befindet (bzw. unter WebSphere 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. Ändern Sie unter WebSphere MQ for AIX, WebSphere MQ for HP-UX oder WebSphere MQ for 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 auf den WebSphere 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 WebSphere 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* wird auf MQXCC_SUPPRESS_FUNCTION gesetzt, sodass die Nachricht, die verarbeitet wird, in die Warteschlange für nicht zustellbare Nachrichten eingereiht und nicht in die Zielwarteschlange eingereiht wird. In MQCXP wird ein Rückkopplungscode zurückgegeben.*Feedback* 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

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

Kompilieren Sie 'amqsxrma' mit folgenden Befehlen:

Unter AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

Unter HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -IMQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -LMQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

unter Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

Unter Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm

 -lsocket
-lnsl -ldl
```

Unter Windows

WebSphere MQ stellt jetzt die Bibliothek 'mqm' mit Clientpaketen und Serverpaketen bereit, sodass im folgenden Beispiel `mqm.lib` anstelle von `mqmvx.lib`verwendet wird:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Allgemeine Informationen zum Schreiben und Kompilieren von Kanalexits finden Sie unter „Kanalexitprogramme schreiben” auf Seite 425

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 veranschaulichte Funktionen” auf Seite 102 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.

UNIX-, Linux- und Windows-Systeme

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 22 auf Seite 158 veranschaulicht die gemeinsame Verwendung der Request- und Inquire-Beispielprogramme.

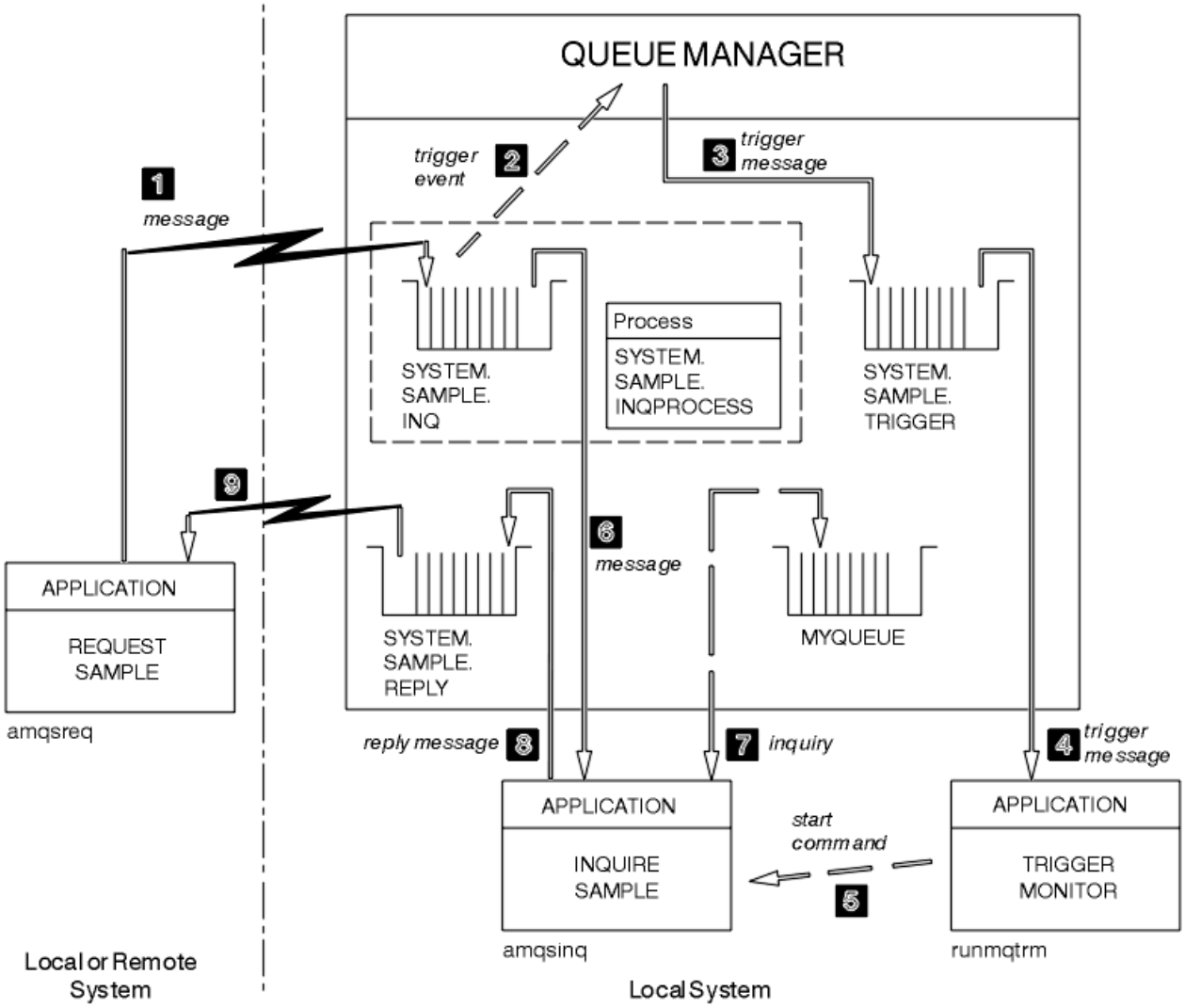


Abbildung 22. Request- und Inquire-Beispielprogramme mittels Triggering ausführen

In Abbildung 22 auf Seite 158 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 22 auf Seite 158 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 159 ein.

Weitere Informationen zum Triggering finden Sie unter „IBM WebSphere MQ-Anwendungen durch Auslöser starten" auf Seite 350.

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.

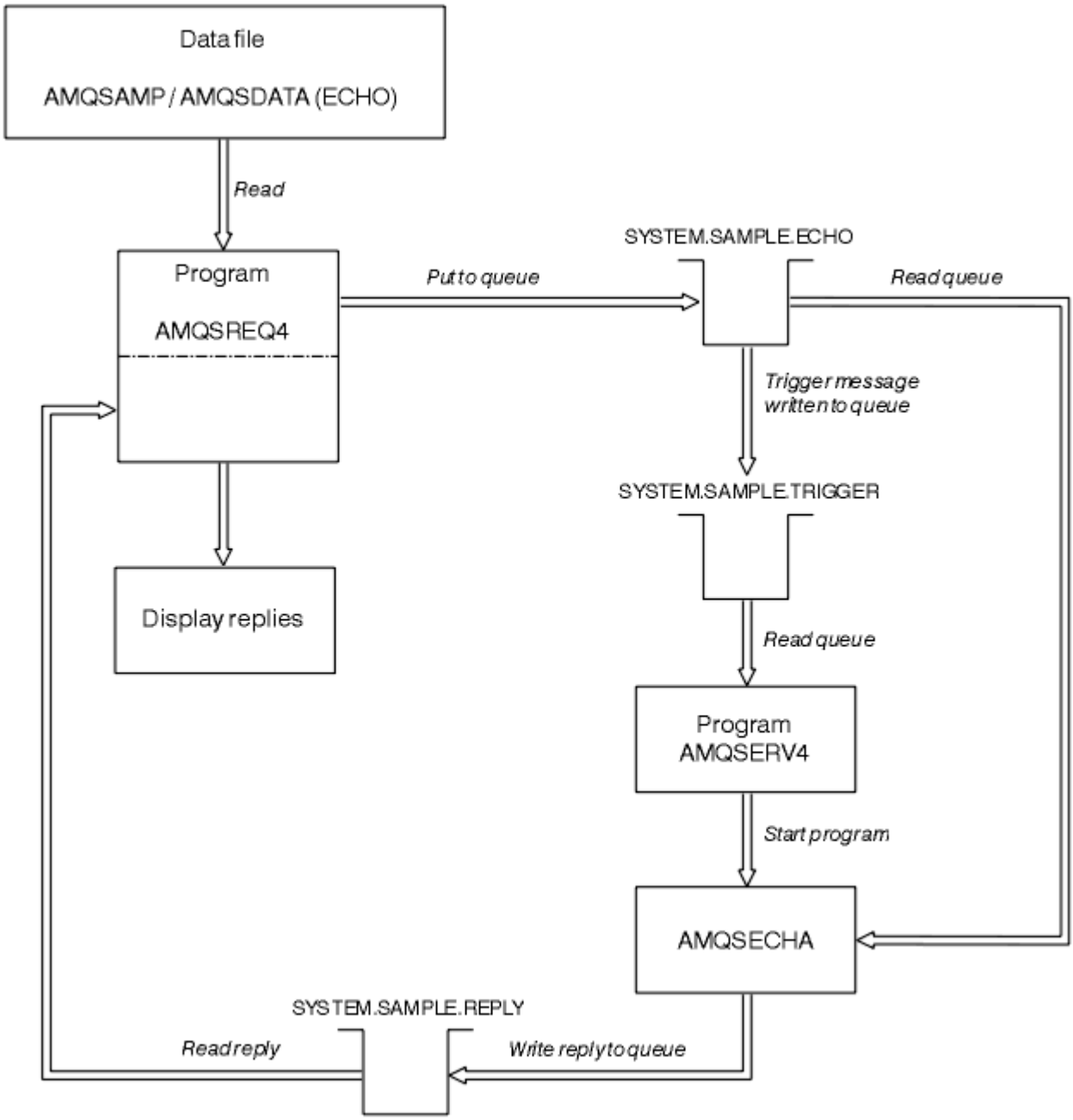


Abbildung 23. Ablaufdiagramm des IBM i Client/Server (Echo)-Beispielprogramms

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 veranschaulichte Funktionen” auf Seite 102.

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:

- Starten Sie auf UNIX-, Linux- und Windows-Systemen das Programm **runmqtrm** in einer Sitzung und anschließend das Programm 'amqsreq' in einer anderen Sitzung.
- 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.

SSL/TLS-Beispielprogramm

AMQSSLC ist ein C-Beispielprogramm, das veranschaulicht, wie die MQCNO- und MQSCO-Strukturen verwendet werden, um SSL/TLS-Clientverbindungsinformationen im MQCONNX-Aufruf bereitzustellen. Damit kann eine MQI-Clientanwendung während ihrer Ausführung die Definition ihres Clientverbindungskanals 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.

SSL/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 SSL/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.

SSL/TLS-Beispielprogramm ausführen

Um das SSL/TLS-Beispielprogramm auszuführen, müssen Sie zuerst Ihre SSL- oder 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 Setting up a key repository on UNIX, Linux, and Windows systems.
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 unter Creating a self-signed personal certificate on UNIX, Linux, and Windows systems.
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 unter Extrahieren des öffentlichen Teils eines selbst signierten Zertifikats aus einem Schlüsselrepository auf UNIX-, Linux -und Windows -Systemen und Hinzufügen eines CA-Zertifikats (oder des öffentlichen Teils eines selbst signierten Zertifikats) in ein Schlüsselrepository auf UNIX-, Linux -oder Windows -Systemen.
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(NULL_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:\Program Files\IBM\WebSphere MQ\clientkey" -s NULL_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 SSL CipherSpec NULL_SHA
Using SSL key repository stem c:\Program Files\IBM\WebSphere 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-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 veranschaulichte Funktionen” auf Seite 102.

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*. Die letzten beiden Felder sind numerisch, sodass das Programm Ersatzzeichen erstellt, die in einer MQTMC2-Struktur für UNIX, Linux und Windows-Systeme verwendet werden.

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). In UNIX-, Linux- und Windows-Systemen wird das Feld *EnvData* als Erweiterung für die aufrufende Befehlsfolge verwendet.

Am Ende schließt das Programm die Initialisierungswarteschlange.

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: Bei der Funktion in diesem Beispiel handelt es sich um einen Teil der im Programm **runmqtrm** bereitgestellten Auslösefunktion.

Konstruktion des Auslöseservers

Die Funktionsweise des Auslöseservers ähnelt der des Auslösemonitors, nur gilt beim Auslöseserver Folgendes:

- Er lässt MQAT_CICS- und MQAT_OS400-Anwendungen zu.
- Bei CICS-Anwendungen ersetzt er *EnvData* (beispielsweise zur Angabe der CICS-Region) über die Auslösenachricht im Befehl 'STRCICSUSR'.
- Er öffnet die Initialisierungswarteschlange für die gemeinsam genutzte Eingabe, sodass viele Auslöseserver gleichzeitig ausgeführt werden können.

Anmerkung: Programme die von AMQSERV4 gestartet werden, dürfen keine MQDISC-Aufrufe verwenden, da der Auslöseserver sonst beendet wird. Wenn über AMQSERV4 gestartete Programme den MQCONN-Aufruf verwenden, wird der Ursachencode MQRC_ALREADY_CONNECTED zurückgegeben.

TUXEDO; Beispielprogramme

Diese Abschnitte enthalten Informationen zu den Put- und Get-Beispielprogrammen für TUXEDO sowie zum Erstellen der Serverumgebung für TUXEDO.

Sie müssen vor dem Ausführen dieser Beispielprogramme zunächst die Serverumgebung erstellen.

Anmerkung: In diesem Abschnitt wird durchgängig der umgekehrte Schrägstrich (\) verwendet, um lange Befehle über mehr als eine Zeile zu trennen. Dieser Backslash gehört nicht zum Befehl und darf daher nicht eingegeben werden. Geben Sie die einzelnen Befehle jeweils in eine einzige Zeile ein.

Serverumgebung erstellen

In diesen Abschnitten finden Sie Informationen zum Erstellen der Serverumgebung für WebSphere MQ auf verschiedenen Plattformen.

Es wird davon ausgegangen, dass eine funktionierende TUXEDO-Umgebung vorhanden ist.

Erstellen der Serverumgebung für WebSphere MQ for AIX (32 Bit)

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 WebSphere 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' Folgendes 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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for AIX (64 Bit)

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 WebSphere MQ installiert ist:

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH□inc -I /<APPDIR> -L MQ_INSTALLATI□
ON_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' Folgendes 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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for Solaris (32 Bit)

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

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 der TUXEDO-Datei 'udataobj/RM' Folgendes hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
```

```
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a
```

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  \

$ buildclient -o dogets  -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/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 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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for Solaris (64 Bit)

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

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 der TUXEDO-Datei 'udataobj/RM' Folgendes hinzu:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a
```

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  \

$ buildclient -o dogets  -f 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 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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for HP-UX (32 Bit)

Anmerkung: Die 32-Bit-Version der TUXEDO-Serverumgebung kann nur auf Itanium-Plattformen erstellt werden.

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

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' Folgendes 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' 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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for HP-UX (64 Bit)

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

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 der TUXEDO-Datei 'udataobj/RM' Folgendes hinzu:

   Für die Plattform HP-UX IA64 (IPF):

   ```
   MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
   MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
   /opt/tuxedo/lib/libtux.sl
   ```

   Anmerkung: Die zusammen mit der Plattform HP-UX IA64 (IPF) bereitgestellten WebSphere 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
   ```

   Für die Plattform HP-UX IA64 (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
```

Sie können jetzt mit den Programmen 'doputs' und 'dogets' Nachrichten in eine Warteschlange einreihen bzw. aus einer Warteschlange abrufen.

Erstellen der Serverumgebung für WebSphere MQ for Windows (32 Bit)

Anmerkung: Ändern Sie die Angaben in spitzen Klammern (<>) in die folgenden Verzeichnispfade:

<MQMDIR>	Der Verzeichnispfad, der bei der Installation von WebSphere MQ angegeben wurde, z. B. `g:\Program Files\IBM\WebSphere MQ`
<TUXDIR>	Der Verzeichnispfad, der bei der Installation von TUXEDO angegeben wurde (Beispiel: `f:\tuxedo`)
<APPDIR>	in den Verzeichnispfad, der für die Beispielanwendung verwendet werden soll; Beispiel: `f:\tuxedo\apps\mqapp`

So erstellen Sie die Serverumgebung und die Beispiele:

1. Erstellen Sie ein Anwendungsverzeichnis, in dem die Beispielanwendung erstellt werden soll; Beispiel:

```
f:\tuxedo\apps\mqapp
```

2. Kopieren Sie die Beispieldateien aus dem WebSphere 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 24 auf Seite 173), 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' folgende Zeile hinzu:

```
MQSeries_XA_RMI;MQRMIXASwitchDynamic;
   <MQMDIR>\tools\lib\mqmxa.lib <MQMDIR>\tools\lib\mqm.lib
```

Dabei wird < MQMDIR > wie im vorherigen Beispiel dargestellt ersetzt. Dieser neue Eintrag ist hier zwar auf zwei Zeilen verteilt, er muss der Datei jedoch in einer einzigen Zeile hinzugefügt werden.

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 folgenden Befehl ein:

```
crdl -z <APPDIR>\TLOG
```

Dabei wird <*APPDIR*> ersetzt.

8. Setzen Sie das aktuelle Verzeichnis auf <APPDIR> und rufen Sie die Beispiel-Makefile (amqstxmn.mak) als eine externe Projekt-Makefile auf, bei Microsoft Visual C++ beispielsweise mit dem folgenden Befehl:

```
msvc amqstxmn.mak
```

Wählen Sie **build**, um alle Beispielprogramme zu erstellen.

```
*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 24. Beispiel für die Datei 'ubbstxcn.cfg' für WebSphere MQ for Windows

Anmerkung: Ändern Sie die Verzeichnisnamen und -pfade entsprechend Ihrer Installation. Außerdem müssen Sie den Namen MYQUEUEMANAGER in den Namen des Warteschlangenmanagers ändern, zu dem eine Verbindung hergestellt werden soll. Alle weiteren Informationen, die hinzugefügt werden müssen, stehen in spitzen Klammern (<>).

Die Beispieldatei 'ubbconfig' für WebSphere MQ for Windows ist in Abbildung 24 auf Seite 173 aufgeführt. Sie wird mit dem Namen 'ubbstxcn.cfg' im Beispielverzeichnis von WebSphere MQ bereitgestellt.

Die bereitgestellte Beispiel-Makefile (siehe Abbildung 25 auf Seite 174) für WebSphere MQ for Windows hat den Namen 'ubbstxmn.mak' und wird im Beispielverzeichnis von WebSphere MQ gespeichert.

```
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 25. TUXEDO-Beispiel-Makefile für WebSphere MQ for Windows

Erstellen der Serverumgebung für WebSphere MQ for Windows (64 Bit)

Anmerkung: Ändern Sie die Angaben in spitzen Klammern (<>) in die folgenden Verzeichnispfade:

<MQMDIR>	Der Verzeichnispfad, der bei der Installation von WebSphere MQ angegeben wurde, z. B. `g:\Program Files\IBM\WebSphere MQ`
<TUXDIR>	Der Verzeichnispfad, der bei der Installation von TUXEDO angegeben wurde (Beispiel: `f:\tuxedo`)
<APPDIR>	in den Verzeichnispfad, der für die Beispielanwendung verwendet werden soll; Beispiel: `f:\tuxedo\apps\mqapp`

So erstellen Sie die Serverumgebung und die Beispiele:

1. Erstellen Sie ein Anwendungsverzeichnis, in dem die Beispielanwendung erstellt werden soll; Beispiel:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Kopieren Sie die Beispieldateien aus dem WebSphere 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 26 auf Seite 176), 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
```

   Dabei muss <MQMDIR> entsprechend ersetzt werden. Dieser neue Eintrag ist hier zwar auf zwei Zeilen verteilt, er muss der Datei jedoch in einer einzigen Zeile hinzugefügt werden.
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 folgenden Befehl ein:

```
crdl -z <APPDIR>\TLOG
```

   Dabei wird <*APPDIR*> wie im vorherigen Beispiel veranschaulicht ersetzt.
8. Setzen Sie das aktuelle Verzeichnis auf <APPDIR> und rufen Sie die Beispiel-Makefile (amqstxmn.mak) als eine externe Projekt-Makefile auf, bei Microsoft Visual C++ beispielsweise mit dem folgenden Befehl:

```
msvc amqstxmn.mak
```

   Wählen Sie **build**, um alle Beispielprogramme zu erstellen.

```
*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 26. Beispiel für die Datei 'ubbstxcn.cfg' für WebSphere MQ for Windows

Anmerkung: Ändern Sie die Verzeichnisnamen und -pfade entsprechend Ihrer Installation. Außerdem müssen Sie den Namen MYQUEUEMANAGER in den Namen des Warteschlangenmanagers ändern, zu dem eine Verbindung hergestellt werden soll. Alle weiteren Informationen, die hinzugefügt werden müssen, stehen in spitzen Klammern (<>).

Die Beispieldatei 'ubbconfig' für WebSphere MQ for Windows ist in Abbildung 26 auf Seite 176 aufgeführt. Sie wird mit dem Namen 'ubbstxcn.cfg' im Beispielverzeichnis von WebSphere MQ bereitgestellt.

Die bereitgestellte Beispiel-Makefile (siehe Abbildung 27 auf Seite 177) für WebSphere MQ for Windows hat den Namen 'ubbstxmn.mak' und wird im Beispielverzeichnis von WebSphere MQ gespeichert.

```
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 27. TUXEDO-Beispiel-Makefile für WebSphere MQ for Windows

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: Sie müssen den Warteschlangenmanager ***vor*** dem Start von TUXEDO 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.

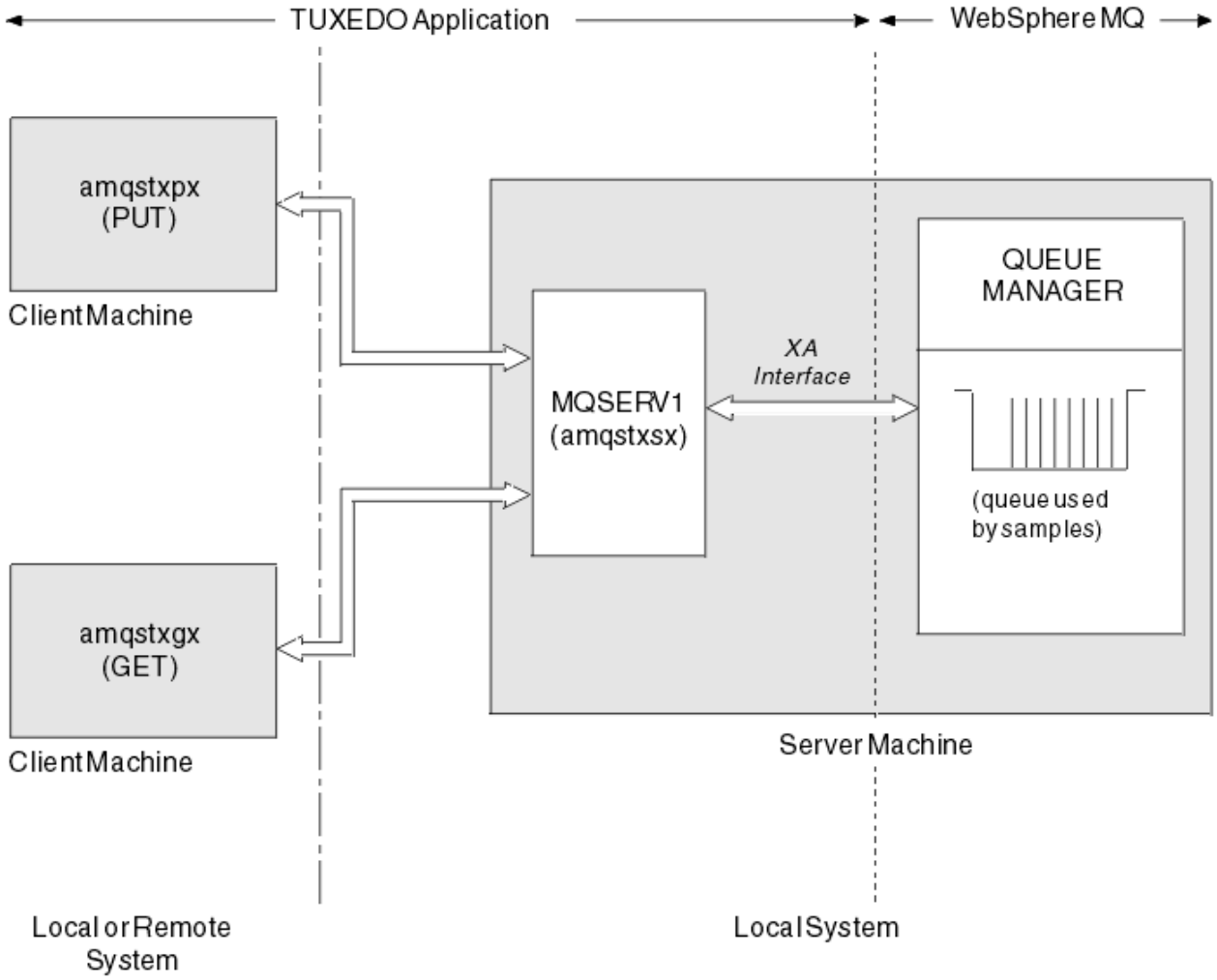


Abbildung 28. Zusammenspiel der TUXEDO-Beispiele

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.

Das Get-Beispielprogramm für TUXEDO

Mit diesem Beispiel können Sie Nachrichten im Stapelbetrieb aus einer Warteschlange abrufen.

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:

- `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.

Verwenden des SSPI-Sicherheitsexits auf Windows-Systemen

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 WebSphere MQ for Windows im Ordner *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* installiert. Beispiel: `C:\Program Files\IBM\WebSphere 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 'DOMÄNE\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 in `C:\Program Files\IBM\WebSphere 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 /I<path_to_Microsoft_platform_SDK\include>
/I<path_to_WebSphere 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 WebSphere 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 WebSphere 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.

Entwerfen

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 WebSphere MQ anhand der Eigenschaft **CLWLPRTY** (zur Priorisierung bei der Clusterauslastung) der einzelnen Warteschlangeninstanzen die Instanzen, an die Nachrichten gesendet werden sollen. WebSphere 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 WebSphere MQ wird die aktualisierte Eigenschaft **CLWLPRTY** einer Clusterwarteschlange an alle betreffenden Warteschlangenmanager im Cluster weitergegeben. Zum 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 WebSphere 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 für die Überwachung der Clusterwarteschlange hängt von dem zugrunde liegenden WebSphere MQ-System und der Clustertechnologie ab. 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 WebSphere MQ-Ressourcen möglich ist. Zum 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 WebSphere 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.
- WebSphere 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

Damit das Beispielprogramm zur Überwachung von Clusterwarteschlangen ausgeführt werden kann, müssen Sie den Warteschlangenmanager so konfigurieren, dass er eingehende Verbindungsanforderungen von Anwendungen, die im Clientmodus aktiv sind, sicher akzeptieren kann.

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 können Sie diese Warteschlange beispielsweise 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 WebSphere 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 **RUNMQSC**-Beispiel können Sie das Beispiel für die Überwachung von Clusterwarteschlangen als IBM WebSphere MQ-Service definieren.

   ```
   define service(AMQSCLM) +
       descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
       control(qmgr) +
       servtype(server) +
       startcmd('<Install Root>\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')
   ```

   Dabei steht <Install Root> für die Position Ihrer Installation.

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 der Dateiname einer Datei, die eine Liste mit Warteschlangennamen oder Masken für Warteschlangennamen enthält, die überwacht werden sollen. 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 von Umlaufnachrichten gespeichert werden < fn> Für jede Kombination aus Warteschlangenmanager und Warteschlange wird eine Berichtsdatei generiert, die auf eine bestimmte Größe begrenzt ist. Die Protokollfunktion schreibt immer in dieselbe Datei, behält aber auch die beiden vorherigen Versionen der Datei. < /fn> Berichtsdateien bei.
-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.

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 WebSphere 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 WebSphere 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 Start.

- Beim Beenden.
- Wenn eine Warteschlange als **ACTIVE** oder **INACTIVE** gekennzeichnet wird.
- Wenn Nachrichten aus einer inaktiven Warteschlange in eine aktive Instanz bzw. in aktive Instanzen eingereiht werden.

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 diesen Fällen 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, Alternativ bleibt sie ACTIVE **CLWLPRTY**> =1, da sie mindestens einen Konsumenten hat.

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 haben Konsumenten (**IPPROCS**> = 1), sind jedoch auf diesen Warteschlangenmanagern inaktiv (**CLWLPRTY**= 0), da AMQSCLM diese fernen Instanzen nicht überwacht.

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.

Das Beispielprogramm für die Suche nach Verbindungsendpunkten (CEPL)

Das IBM WebSphere MQ-Beispielprogramm für die Suche nach Verbindungspunkten (CEPL) stellt ein einfaches, aber leistungsfähiges Exitmodul bereit, mit dem WebSphere MQ-Benutzer Verbindungsdefinitionen aus einem LDAP-Respository wie beispielsweise Tivoli Directory Server abrufen können.

Damit CEPL verwendet werden kann, muss der Tivoli Directory Server v6.3-Client installiert sein.

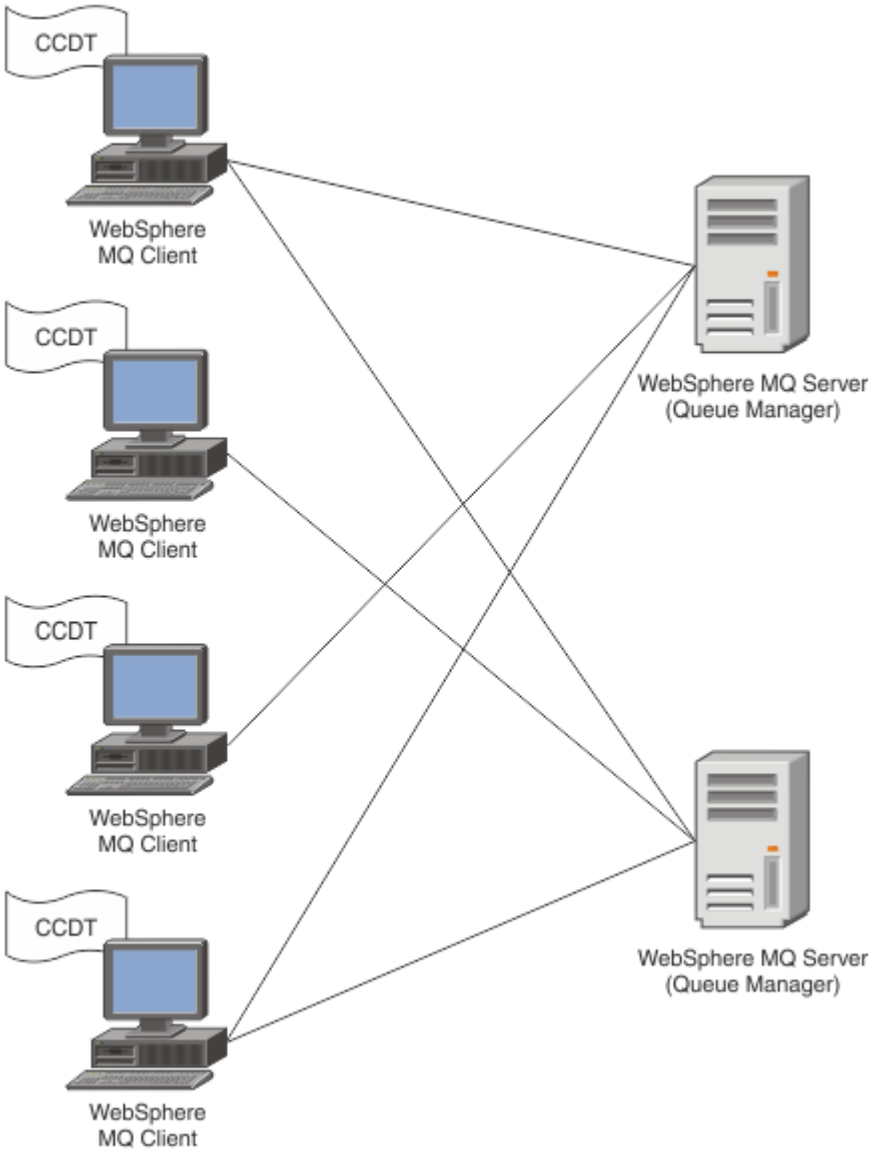
Für die Verwendung dieses Beispielprogramms sind praktische Erfahrungen mit der Verwaltung von WebSphere MQ auf den unterstützten Plattformen erforderlich.

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 WebSphere 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 WebSphere 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



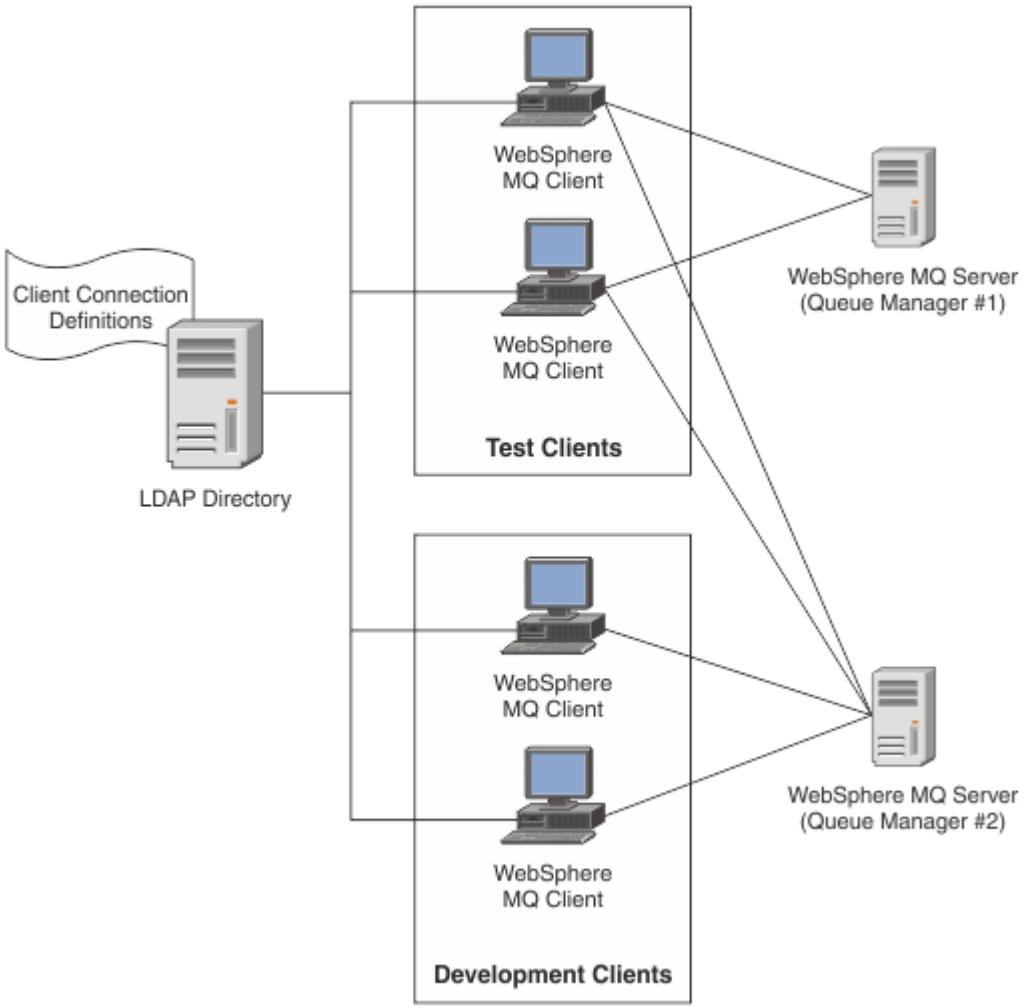
verteilt werden.

Die CCDT-Datei muss an jeden WebSphere 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 WebSphere 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 WebSphere MQ bereitgestellten Funktionen nutzen möchten.
- Neu- oder Bestandskunden, die ihr eigenes Messaging-Modell direkt innerhalb der bestehenden LDAP-Geschäftsarchitekturen verwenden oder optimieren möchten.

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 WebSphere MQ für die Suche nach Verbindungsendpunkten ist die folgende Software erforderlich:

- IBM WebSphere MQ V7.0 oder höher
- Tivoli Directory Server V6.3-Client oder höher

Unterstützte Betriebssysteme:

1. Windows (XP/2003/2008)
2. Solaris (SPARC und x86-64)
3. AIX
4. Linux
   - RHEL V4 und V5 unter System p
   - SUSE V9 und V10 unter System p
   - RHEL V4 und V5 System x32 Bit und x64 Bit
   - SUSE V9 und V10 System x32 Bit und x64 Bit
5. HP IA64

Anmerkung: Für die Plattformen z/OS, i/5 und HP PARISC ist das Beispiel nicht verfügbar.

Installieren und konfigurieren

Dieser Abschnitt enthält Informationen zum Installieren und Konfigurieren des Exitmoduls und des Schemas für Verbindungsendpunkte.

Exitmodul installieren

Das Exitmodul wird bei der Installation von WebSphere MQ im Verzeichnis `tools/samples/c/preconnexit/bin` installiert. Auf 32-Bit-Plattformen muss das Exitmodul in `exit/<install name>/` kopiert werden, damit es verwendet werden kann. Auf 64-Bit-Plattformen muss das Exitmodul in das Verzeichnis 'exit64/<Installationsname>/' kopiert werden, bevor 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 den folgenden Befehl ein, um das Schema im ITDS-Speicher zu installieren; dabei handelt es sich bei 'LDAP ID' und 'LDAP password' um den Root-DN und das Rootkennwort für den LDAP-Server:

  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

Der Clientkonfigurationsdatei (beispielsweise *mqclient.ini*) muss ein neuer Abschnitt (**PreConnect**) hinzugefügt werden. Der Abschnitt 'PreConnect' enthält die folgenden Schlüsselwörter:

Module: Der Name des Moduls, das den API-Exit-Code enthält. Enthält dieses Feld den vollständigen Pfad des Moduls, wird es unverändert verwendet; andernfalls werden die Ordner *exit* oder *exit64* der WebSphere MQ-Installation durchsucht.

Function: Name des funktionalen Einstiegspunkts in die Bibliothek mit dem PreConnect-Exit-Code. Die Funktionsdefinition basiert auf dem MQ_PRECONNECT_EXIT-Prototyp.

Data: URI des LDAP-Repositorys, das Kanaldefinitionen enthält.

Das folgende Snippet veranschaulicht die erforderlichen Änderungen an der Datei *mqclient.ini*:

```
PreConnect:
  Module=amqlcelp
  Function=PreConnectExit
  Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
  Sequence=1
```

Überblick über den Exit und das Schema

Dieser Abschnitt enthält Informationen zur Syntax und zu den Parametern, mit denen eine Verbindung zu einem Warteschlangenmanager hergestellt wird.

WebSphere MQ v7.5 definiert die folgende Syntax für einen Eingangspunkt in einem Exitmodul.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP  pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Bei der Ausführung des MQCONN/X-Aufrufs lädt der WebSphere 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. Der Parameter ist NULL, wenn der Exit mit MQXR_TERM aufgerufen wird.

ppConnectOpts

Typ: ppConnectOpts Ein-/Ausgabe

Optionen, mit denen die Aktion des MQCONNX-Aufrufs gesteuert wird. Hierbei handelt es sich um einen Verweis auf eine MQCNO-Struktur für Verbindungsoptionen, mit denen die Aktion des API-Aufrufs MQCONN gesteuert wird. Der Parameter ist NULL, wenn der Exit mit MQXR_TERM aufgerufen wird. Der MQI-Client stellt immer eine MQCNO-Struktur für den Exit bereit, auch wenn sie von der Anwendung ursprünglich nicht bereitgestellt wurde. Stellt eine Anwendung eine MQCNO-Struktur bereit, erstellt der Client ein Duplikat davon, das an den Exit übergeben und dort geändert wird. Der Client bleibt Eigner der MQCNO-Struktur. Eine MQCD-Struktur, auf die in der MQCNO-Struktur verwiesen wird, hat Vorrang vor allen Verbindungsdefinitionen, die über das Array bereitgestellt werden. Der Client stellt mithilfe der MQCNO-Struktur eine Verbindung zum Warteschlangenmanager her, während die anderen Strukturen ignoriert werden.

pCompCode

Typ: PMQLONG Ein-/Ausgabe

Beendigungscode. Verweis auf eine MQLONG-Struktur, die den Beendigungscode des Exits empfängt. Folgende Werte sind zulässig:

MQCC_OK: Erfolgreich beendet

MQCC_WARNING: Warnung (nicht vollständig beendet)

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.

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 */
};
```

Beispielcode zum Erstellen des Exits für die Suche nach Verbindungsendpunkten

Beispiel-Code-Snippets für das Kompilieren der Quelle unter Windows und auf verteilten Plattformen.

Quelle kompilieren

Sie können die Quelle mit beliebigen LDAP-Clientbibliotheken (beispielsweise den IBM Tivoli Directory Server 6.3-Clientbibliotheken) kompilieren. In dieser Dokumentation wird von einer Verwendung der Tivoli Directory Server 6.3-Clientbibliotheken ausgegangen.

Anmerkung: Die Preconnect-Exitbibliothek wurde mit den folgenden LDAP-Servern getestet:

- IBM Tivoli Directory Server 6.3
- Novell eDirectory V8.2

Das folgende Code-Snippet veranschaulicht die Kompilierung der Exits unter Windows und auf anderen verteilten Plattformen:

Kompilieren des Exits auf der Windows-Plattform

Mit dem folgenden Snippet können Sie die Exitquelle unter Windows 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: Bei Verwendung von IBM Tivoli Directory Server 6.3-Clientbibliotheken, die mit dem Compiler von Microsoft Visual Studio 2005 oder höher kompiliert wurden, erhalten Sie eventuell Warnmeldungen, wenn Sie die IBM Tivoli Directory Server 6.3-Clientbibliotheken mit dem Compiler von Microsoft Visual Studio 2003 kompilieren.

Exit auf anderen verteilten Plattformen kompilieren

Mit dem folgenden Snippet können Sie die Exitquelle auf anderen verteilten Plattformen (beispielsweise Linux) kompilieren. Einige Compileroptionen können von verteilter Plattform zu verteilter Plattform variieren.

```
#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 sowohl statische als auch DLL-Dateien, aber nur eine Art von Bibliotheken kann 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)
```

Aufruf des Exitmoduls

Das Preconnect-Exitmodul kann mit drei verschiedenen Ursachencodes aufgerufen werden. In diesem Abschnitt werden die einzelnen Ursachencodes eingehender beschrieben.

MQXR_INIT

Der Exit wird mit dem Ursachencode MQXR_INIT aufgerufen, um eine Verbindung zu einem LDAP-Server zu initialisieren und herzustellen.

Das Feld *pExitDataPtr* der MQNXP-Struktur wird vor dem *MQXR_INIT*-Aufruf 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 zum LDAP-Server her. Die daraus resultierenden LDAP-API-Kennungen werden ebenfalls in dieser Struktur gespeichert.

MQXR_PRECONNECT

Das Exitmodul wird mit dem Ursachencode MQXR_PRECONNECT zum Abrufen der Kanaldefinitionen aus einem LDAP-Server aufgerufen.

Der Exit durchsucht den LDAP-Server auf Kanaldefinitionen, die mit dem angegebenen Filter übereinstimmen. Enthält der Parameter *QMgrName* 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 übereinstimmen.

Hat der Parameter *QMgrName* den Wert '*' oder ist er leer (' '), werden bei der Suche alle Kanaldefinitionen zurückgegeben, bei denen das Verbindungsendpunktattribut *ibm-amqIsClientDefault* 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. Der Exit trennt dabei die Verbindung zum LDAP-Server, wobei der vom Exit zugeordnete und verwaltete Speicher freigegeben wird. Dazu gehört auch die Struktur *MQNLDAPCTX*, das Verweisarray und jede MQCD-Struktur, auf die verwiesen wird. Alle anderen Felder werden auf den jeweiligen Standardwert gesetzt. Die Exitparameter *pQMgrName* und *ppConnectOpts* werden bei *MQXR_TERM* nicht verwendet und können auf null gesetzt sein.

LDAP-Schemas

Clientverbindungsdaten werden in einem globalen Repository gespeichert, dem LDAP-Verzeichnis (Lightweight Directory Access Protocol). Aus einem LDAP-Verzeichnis rufen WebSphere MQ-Clients die Verbindungsdefinitionen ab. Die Struktur der WebSphere 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 WebSphere MQ-Eigenschaften finden Sie unter LDAP-Attribute.

LDAP-Attribute

Definierte LDAP-Attribute sind WebSphere MQ-spezifisch und entsprechen direkt den Clientverbindungseigenschaften.

Verzeichniszeichenfolgeattribute für WebSphere MQ-Clientkanäle

Die folgende Tabelle enthält eine Beschreibung der Zeichenfolgeattribute und ihre Zuordnung zu WebSphere 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 21. Verzeichniszeichenfolgeattribute für WebSphere MQ-Clientkanäle

LDAP-Attribut	Beschreibung	WebSphere 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
ibm-amqLocalAddress	Die lokale Kommunikationsadresse des Kanals.	LOCLADDR
ibm-amqModeName	Der LU-6.2-Modusname.	MODENAME
ibm-amqPassword	Das Kennwort, das verwendet werden kann.	KENNWORT
ibm-amqQueueManagerName	Der Name des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern, zu denen eine WebSphere MQ-Clientanwendung eine Verbindung anfordern kann.	QMNAME
ibm-amqSecurityExitUserData	Die Benutzerdaten, die an den Sicherheitsexit übergeben werden.	SCYDATA
ibm-amqSecurityExitName	Der Name des Exitprogramms, das vom Kanalsicherheitsexit ausgeführt werden soll.	SCYEXIT
ibm-amqSslCipherSpec	Eine einzelne CipherSpec für eine SSL-Verbindung.	SSLCIPH
ibm-amqSslPeerName	Überprüft den definierten Namen (DN) des Zertifikats vom Peer-Warteschlangenmanager oder Peer-Client am anderen Ende eines WebSphere MQ-Kanals.	SSLPEER

Tabelle 21. Verzeichniszeichenfolgeattribute für WebSphere MQ-Clientkanäle (Forts.)		
LDAP-Attribut	**Beschreibung**	**WebSphere MQ-Eigenschaft**
ibm-amqTransactionProgram-Name	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 WebSphere 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 22. Verzeichnisganzzahlattribute für WebSphere MQ-Clientkanäle		
LDAP-Attribut	**Beschreibung**	**WebSphere 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
ibm-amqMaximumMessage-Length	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 WebSphere MQ-Clientverbindungen

Dieses boolesche Attribut wird keiner WebSphere MQ-Eigenschaft zugeordnet. Als Syntax für dieses Attribut wird ein boolescher Wert verwendet.

Tabelle 23. Boolesches Attribut für WebSphere MQ-Clientverbindungen	
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 WebSphere MQ-Clientkanal

WebSphere MQ-Eigenschaften werden im LDAP-Verzeichnis als Listenattribute mit durch Kommas getrennten Einzelwerten gespeichert. Die Attribute werden auf dieselbe Weise wie die anderen Ver-

zeichniszeichenfolgeattribute definiert. Die folgende Tabelle enthält eine Beschreibung der Listenattribute und ihre Zuordnung zu WebSphere MQ-Eigenschaften.

Tabelle 24. Listenattribute für WebSphere MQ-Clientkanal

LDAP-Attribut	Beschreibung	WebSphere 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

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 ein oder mehrere Platzhalter enthält, das Verhalten des Attributs in einem Suchfilter angibt.

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.

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.

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.

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.

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.

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.

ibm-amqQueueManagerName, LDAP-Attribut
Dieses Attribut gibt den Namen des Warteschlangenmanagers bzw. der Gruppe von Warteschlangenmanagern an, zu denen eine WebSphere 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.

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.

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.

ibm-amqSslCipherSpec, LDAP-Attribut
Dieses LDAP-Attribut gibt eine einzelne CipherSpec für eine SSL-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.

ibm-amqSslPeerName
Mit diesem LDAP-Attribut wird der definierte Name (DN) des Zertifikats vom Peer-Warteschlangenmanager oder Peer-Client am anderen Ende eines WebSphere 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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. Die Angabe `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.

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.

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.

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.

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.

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.

Anwendung zur Warteschlangensteuerung schreiben

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:

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„IBM WebSphere MQ-Anwendungen entwerfen” auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„WebSphere MQ-Beispielprogramme” auf Seite 100
In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

„Clientanwendungen schreiben” auf Seite 374
Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„Web-Services in WebSphere MQ verwenden” auf Seite 1004
Sie können IBM WebSphere MQ-Anwendungen für Web-Services mithilfe des IBM WebSphere MQ-Transports für SOAP oder der IBM WebSphere MQ-Bridge für HTTP entwickeln.

„IBM WebSphere MQ-Anwendung erstellen” auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

„Programmfehler behandeln” auf Seite 581
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.

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

In WebSphere MQ für Windows und WebSphere MQ unter UNIX and Linux wird außerdem Folgendes bereitgestellt:

- Aufrufe, über die Programme in WebSphere MQ für Windows und WebSphere 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 für verteilte Plattformen” auf Seite 101.
- Beispielquellcode und ausführbaren Beispielcode für die Verbindung zu externen Transaktionsmanagern

Unter den folgenden Links finden Sie weitere Informationen zur MQI:

- „MQI-Aufrufe” auf Seite 208
- „Synchronisationspunktaufrufe” auf Seite 209
- „Datenkonvertierung, Datentypen, Datendefinitionen und Strukturen” auf Seite 209
- „IBM WebSphere MQ-Stubprogramme und -Bibliotheksdateien” auf Seite 210
- „Parameter, die in allen Aufrufen verwendet werden” auf Seite 214
- „Puffer angeben” auf Seite 215
- „Signalverarbeitung unter UNIX and Linux” auf Seite 216

Zugehörige Konzepte

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

MQI-Aufrufe

Dieser Abschnitt enthält Informationen zu den MQI-Aufrufen.

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 schreiben, werden diese Aufrufe nicht benötigt. 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
Verwenden Sie diese Aufrufe, wenn WebSphere MQ als Koordinator einer Arbeitseinheit fungiert. Mit MQBEGIN wird die Arbeitseinheit gestartet, Mit MQCMIT und MQBACK wird sie beendet, wobei die im Verlauf der Arbeitseinheit vorgenommenen Aktualisierungen entweder festgeschrieben oder rückgän-

gig gemacht werden. 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 WebSphere MQ -Aufrufe auf Windows-, UNIX-und Linux -Plattformen

In den folgenden Produkten werden die Aufrufe MQCMIT und MQBACK bereitgestellt:

- IBM WebSphere MQ for Windows
- IBM WebSphere 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. Um Änderungen in der CICS -Umgebung festzuschreiben und zurückzusetzen, verwenden Sie Befehle wie EXEC CICS SYNCPOINT und EXEC CICS SYNCPOINT ROLLBACK.

Datenkonvertierung, Datentypen, Datendefinitionen und Strukturen

Dieser Abschnitt enthält Informationen zur Datenkonvertierung, zu Elementardatentypen, zu WebSphere 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 WebSphere 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 442.

Elementardatentypen

Für die unterstützten Programmiersprachen stellt die MQI Elementardatentypen oder unstrukturierte Felder bereit.

Eine ausführliche Beschreibung dieser Datentypen finden Sie unter Elementary data types.

WebSphere MQ-Datendefinitionen

Zu den gemeinsam mit WebSphere MQ bereitgestellten Datendefinitionen gehören:

- Definitionen aller WebSphere MQ-Konstanten und -Rückgabecodes
- Definitionen der WebSphere MQ-Strukturen und Datentypen
- Konstantendefinitionen für die Initialisierung der Strukturen
- Funktionsprototypen für jeden Aufruf (nur für PL/I und C)

Eine vollständige Beschreibung der Datendefinitionsdateien von WebSphere MQ finden Sie unter „IBM WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

Strukturen

Strukturen, die zusammen mit den unter „MQI-Aufrufe” auf Seite 208 aufgeführten MQI-Aufrufen verwendet werden, werden in Datendefinitionsdateien für jede unterstützte Programmiersprache bereitgestellt.

Eine Übersicht über die Strukturen finden Sie unter Structure data types summary.

IBM WebSphere 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 „IBM WebSphere MQ-Anwendung erstellen” auf Seite 456. Informationen zum Herstellen einer Verknüpfung mit C++-Bibliotheksdateien finden Sie unter Verwendung von C++*WebSphere MQ C++ verwenden*.

IBM WebSphere MQ for Windows

Unter IBM WebSphere MQ für Windowsmüssen Sie Ihr Programm zusätzlich zu den vom Betriebssystem bereitgestellten Bibliotheksdateien mit den MQI-Bibliotheksdateien verknüpfen, die für die Umgebung bereitgestellt werden, in der Ihre Anwendung ausgeführt wird:

Tabelle 25. 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_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib	Server-TXSeries CICS-Support für C (32 Bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib	Client-TXSeries CICS-Support 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)

Tabelle 25. Bibliotheksdateien für Windows-Anwendungen (Forts.)	
Bibliotheksdatei	**Umgebung**
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 WebSphere MQ installiert ist.

Verwenden Sie für das Kompilieren von .NET-Programmen `amqmdnet.dll`. Weitere Informationen finden Sie unter „WebSphere MQ .NET-Programme kompilieren” auf Seite 630 im Abschnitt „.NET verwenden” auf Seite 594.

Die folgenden Dateien werden aus Gründen der Kompatibilität mit früheren Releases bereitgestellt:

mqic32.lib
mqic32xa.lib

IBM WebSphere MQ für AIX

In IBM WebSphere MQ für AIX müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien, sondern auch noch mit den MQI-Bibliotheksdateien verbinden, die für die Umgebung bereitgestellt werden, in denen die Anwendung ausgeführt wird.

In Nicht-Thread-Anwendungen:

Tabelle 26. Bibliotheksdateien für Nicht-Thread-AIX-Anwendungen	
Bibliotheksdatei	**Umgebung**
libmqm.a	Server für C
libmqic.a & **libmqm.a**	Client für C
libmqmzf.a	Exits für installierbare Services für C

Tabelle 26. Bibliotheksdateien für Nicht-Thread-AIX-Anwendungen (Forts.)	
Bibliotheksdatei	**Umgebung**
libmqmxa.a	Server-XA-Schnittstelle
libmqmxa64.a	Alternative Server-XA-Schnittstelle
libmqcxa.a	Client-XA-Schnittstelle
libmqcxa64.a	Alternative Client-XA-Schnittstelle
libmqmcbrt.o	WebSphere MQ-Laufzeitbibliothek zur Unterstützung von Micro Focus COBOL
libmqmcb.a	Server für COBOL
libmqicb.a	Client für COBOL
libimqc23ia.a	Client für C++
libimqs23ia.a	Server für C++

In einer Thread-Anwendung:

Tabelle 27. Bibliotheksdateien für Thread-AIX-Anwendungen	
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++

IBM WebSphere MQ für HP-UX

In IBM WebSphere MQ für HP-UX müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien, sondern auch noch mit den MQI-Bibliotheksdateien verbinden, die für die Umgebung bereitgestellt werden, in denen die Anwendung ausgeführt wird.

IA64-Plattform (IPF)

In Nicht-Thread-Anwendungen:

Tabelle 28. Bibliotheksdateien für Nicht-Thread-HP-UX-Anwendungen	
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

Tabelle 28. Bibliotheksdateien für Nicht-Thread-HP-UX-Anwendungen (Forts.)	
Bibliotheksdatei	**Umgebung**
libmqmxa64.so	Alternative Server-XA-Schnittstelle
libmqcxa.so	Client-XA-Schnittstelle
libmqcxa64.so	Alternative Client-XA-Schnittstelle
libimqi23ah.so	C + +
libmqmcbrt.o	WebSphere MQ-Laufzeitbibliothek zur Unterstützung von Micro Focus COBOL
libmqmcb.so	Server für COBOL
libmqicb.so	Client für COBOL

In einer Thread-Anwendung:

Tabelle 29. Bibliotheksdateien für Thread-HP-UX-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 + +

IBM WebSphere MQ fürLinux

Unter IBM WebSphere MQ for Linuxmüssen Sie Ihr Programm zusätzlich zu den vom Betriebssystem bereitgestellten Bibliotheksdateien mit den MQI-Bibliotheksdateien verknüpfen, die für die Umgebung bereitgestellt sind, in der Ihre Anwendung ausgeführt wird.

In Nicht-Thread-Anwendungen:

Tabelle 30. 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++

In einer Thread-Anwendung:

Tabelle 31. 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++

IBM WebSphere MQ für Solaris

In IBM WebSphere MQ für Solaris müssen Sie Ihr Programm nicht nur mit den vom Betriebssystem bereitgestellten Bibliotheksdateien, sondern auch noch mit den MQI-Bibliotheksdateien verbinden, die für die Umgebung bereitgestellt werden, in denen die Anwendung ausgeführt wird.

Tabelle 32. 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++

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 WebSphere MQ-Objekt arbeiten kann, muss dem Programm eine eindeutige Kennung bereitgestellt werden, anhand derer es das Objekt identifizieren 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:

- Ursachencodes für alle anderen WebSphere MQ-Plattformen

Puffer angeben

Der Warteschlangenmanager verweist nur bei Bedarf auf Puffer. Ist für einen Aufruf kein Puffer erforderlich oder hat der Puffer eine Länge von null, können Sie einen Nullzeiger auf einen Puffer verwenden.

Zur Größenangabe des Puffers sollte immer die Datenlänge verwendet werden.

Soll in einem Puffer die Ausgabe eines Aufrufs gespeichert werden (beispielsweise die Nachrichtendaten für einen MQGET-Aufruf oder die mit dem MQINQ-Aufruf abgerufenen Attributwerte), wird der Warteschlangenmanager versuchen, einen Ursachencode zurückzugeben, wenn der angegebene Puffer ungültig ist oder es sich bei ihm um einen Nur-Lese-Speicher handelt. Der Warteschlangenmanager wird jedoch unter Umständen nicht immer in der Lage sein, einen Ursachencode zurückzugeben.

Hinweise 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.

Systemaufruf 'fork' unter UNIX and Linux

Bei der Verwendung des Systemaufrufs 'fork' in IBM WebSphere MQ- Anwendungen sollten Sie Folgendes beachten.

Wenn Ihre Anwendung `fork`verwenden möchte, muss der übergeordnete Prozess dieser Anwendung `fork` aufrufen, bevor IBM WebSphere MQ aufgerufen wird, z. B. `MQCONN`, oder ein IBM WebSphere MQ -Objekt mit **`ImqQueueManager`**erstellen.

Wenn Ihre Anwendung nach Ausgabe eines IBM WebSphere MQ-Aufrufs einen untergeordneten Prozess erstellen möchte, muss der Anwendungscode eine `fork()`-Funktion mit `exec()` verwenden, damit es sich bei dem untergeordneten Prozess um eine neue Instanz, nicht um eine exakte Kopie des übergeordneten Prozesses handelt.

Verwendet Ihre Anwendung keine `exec()`-Funktion, wird von dem innerhalb des untergeordneten Prozesses ausgegebenen IBM WebSphere MQ-API-Aufruf der Fehler MQRC_ENVIRONMENT_ERROR zurückgegeben.

Signalverarbeitung unter UNIX and Linux

Dies trifft nicht auf WebSphere MQ for z/OS und WebSphere MQ for Windows zu.

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.

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 WebSphere MQ, das Signal vor dem Ausgeben des MQI-Funktionsaufrufs an einen registrierten Signalhandler zu übergeben.

Thread-Anwendungen
Ein Thread bleibt von einem MQCONN-Aufruf (oder einem MQCONNX-Aufruf) bis zu einem MQDISC-Aufruf mit WebSphere 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 WebSphere MQ jedoch eigene Signalhandler im Anwendungsprozess, solange ein Thread mit WebSphere MQ verbunden ist:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Wenn Sie Multithread-Anwendungen erstellen, gibt es für jedes Signal nur einen prozessübergreifenden Signalhandler. Wenn WebSphere MQ eigene Handler für synchrone Signale definiert, werden für jedes Signal alle zuvor registrierten Handler gespeichert. Nachdem WebSphere MQ eines der aufgeführten Signale verarbeitet hat, versucht WebSphere MQ den Signalhandler aufzurufen, der bei der ersten WebSphere MQ-Verbindung innerhalb des Prozesses gerade aktiv war. Nachdem die Verbindung aller Anwendungsthreads zu WebSphere MQ wieder getrennt wurde, werden die zuvor registrierten Handler wiederhergestellt.

Da Signalhandler von WebSphere 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 WebSphere 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 WebSphere MQ verbunden ist, muss der Signalhandler der Anwendung bei der Verarbeitung des Signals den entsprechenden WebSphere 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 WebSphere MQ einen Signalhandler definiert, muss der vorherige Signalhandler wiederhergestellt werden, bevor die Anwendung die Verbindung zu WebSphere MQ trennt.
- Wenn eine Anwendung vor dem Herstellen einer Verbindung zu WebSphere MQ einen Signalhandler definiert, muss die Verbindung der Anwendung zu WebSphere 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

WebSphere MQ verwendet asynchrone Signale in Thread-Anwendungen nur, wenn es sich um Clientanwendungen handelt.

Zusätzliche Hinweise zu Thread-Client-Anwendungen

WebSphere 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)

Weitere Überlegungen
Bei der Verwendung der UNIX-Signalverarbeitung müssen Sie außerdem noch Folgendes beachten:

Fastpath-Anwendungen (gesicherte Anwendungen)

Fastpath-Anwendungen werden in demselben Prozess wie WebSphere MQ und damit in der Multithread-Umgebung ausgeführt.

In dieser Umgebung werden die synchronen Signale SIGSEGV, SIGBUS, SIGFPE und SIGILL von WebSphere MQ verarbeitet; alle anderen Signale dürfen nicht an die Fastpath-Anwendung übergeben werden, solange sie mit WebSphere 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 222.

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 WebSphere 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.

In der AIX-Umgebung müssen für WebSphere 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 WebSphere MQ andernfalls Aufrufe, die durch ein Signal unterbrochen wurden, unter Umständen nicht abschließen kann.

Benutzerexits und installierbare Services

Benutzerexits und installierbare Services, die als Teil eines WebSphere MQ-Prozesses in einer Multithread-Umgebung ausgeführt werden, unterliegen denselben Einschränkungen wie Fastpath-Anwendungen. Sie sind permanent mit WebSphere 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 WebSphere 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 WebSphere MQ-Anwendung inaktiviert wurden. Sind Upcalls für die WebSphere 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

Um WebSphere MQ-Programmierungsservice zu verwenden, 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:

Nur für z/OS-Stapel, WebSphere MQ for IBM i, WebSphere MQ auf UNIX-Systemen, WebSphere MQ auf Linux-Systemen und WebSphere MQ for Windows unterstützt.

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-Stapelprogramme können nacheinander oder gleichzeitig eine Verbindung zu mehreren Warteschlangenmanagern innerhalb desselben TCB herstellen.

IMS

Die IMS-Steuerregion ist beim Start mit einem oder mehreren Warteschlangenmanagern verbunden. Diese Verbindung wird durch IMS -Befehle gesteuert. 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.

IMS-Programme können nacheinander oder gleichzeitig eine Verbindung zu mehreren Warteschlangenmanagern innerhalb desselben TCB herstellen.

CICS Transaction Server for z/OS und CICS for MVS/ESA

CICS-Programme müssen nichts unternehmen, um eine Verbindung zum Warteschlangenmanager herzustellen, da das CICS-System selbst verbunden ist. Diese Verbindung wird in der Regel während der Initialisierung automatisch hergestellt, Sie können jedoch auch die CKQC-Transaktion verwenden, diedas mit WebSphere MQ for z/OS bereitgestellt wird.

CICS-Tasks können nur zu dem Warteschlangenmanager eine Verbindung herstellen, mit dem die CICS-Region selbst verbunden ist.

Anmerkung: CICS-Programme können auch die entsprechenden MQI-Aufrufe (MQCONN und MQDISC) verwenden, um eine Verbindung herzustellen bzw. zu trennen. Unter Umständen möchten Sie davon Gebrauch machen, damit diese Anwendungen mit möglichst wenig Änderungen am Programmcode in Nicht-CICS-Umgebungen portiert werden können. Diese Aufrufe werden jedoch *immer* in einer CICS -Umgebung erfolgreich ausgeführt. d. h., der Rückgabecode gibt nicht unbedingt Aufschluss über den tatsächlichen Status der Verbindung zum Warteschlangenmanager.

TXSeries für Windows und Open Systems
Diese Programme müssen nichts unternehmen, um eine Verbindung zum Warteschlangenmanager herzustellen, da das CICS-System selbst verbunden ist. Daher wird immer nur jeweils eine Verbindung unterstützt. CICS-Anwendungen müssen einen MQCONN-Aufruf absetzen, um eine Verbindungskennung abzurufen, und vor ihrer Beendigung einen MQDISC-Aufruf ausgeben.

Unter den folgenden Links finden Sie weitere Informationen zum Herstellen bzw. Trennen einer Verbindung zu einem Warteschlangenmanager:

- „Verbindung zu einem Warteschlangenmanager über den MQCONN-Aufruf herstellen” auf Seite 220
- „Verbindung zu einem Warteschlangenmanager über MQCONNX-Aufrufe herstellen” auf Seite 222
- „Programme mit MQDISC von einem Warteschlangenmanager trennen” auf Seite 227

Zugehörige Konzepte
„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Objekte öffnen und schließen” auf Seite 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

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:

- Für IBM WebSphere MQ for z/OS wird in der Stapelumgebung der Standardwarteschlangenmanager im Modul CSQBDEFV angegeben.
- Für IBM WebSphere MQ for Windows-, IBM i-, UNIX- und Linux-Systeme wird der Standardwarteschlangenmanager in der Datei mqs.ini angegeben.

Alternativ können Sie in z/OS MVS -Stapel-, TSO-und RRS-Umgebungen 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 WebSphere 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 bei der Installation von IBM WebSphere MQ for z/OS im Modul CSQQDEFV angegeben.

Für die TXSeries CICS-Umgebung sowie für TXSeries für Windows und AIX muss der Warteschlangenmanager als XA-Ressource in CICS definiert werden.

Eine Verbindung zum Standardwarteschlangenmanager wird mit dem MQCONN-Aufruf hergestellt, indem ein Name eingegeben wird, der aus Leerzeichen besteht oder mit einem Nullzeichen (X'00') beginnt.

Eine Anwendung muss über eine entsprechende Berechtigung verfügen, damit sie erfolgreich eine Verbindung zu einem Warteschlangenmanager herstellen kann. Weitere Informationen finden Sie unter Security.

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 229).

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 WebSphere 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 Websphere MQ-Verbindung von einem Thread hergestellt werden. Alternativ können Sie „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX” auf Seite 225 in Betracht ziehen, um mehrere WebSphere MQ -Verbindungen von einem einzelnen Thread und eine WebSphere MQ -Verbindung von einem beliebigen Thread zuzulassen.[1]

Wenn Ihre Anwendung als Client ausgeführt wird, kann sie innerhalb eines Threads eine Verbindung zu mehreren Warteschlangenmanagern herstellen.

[1] Wenn Sie Multithread-Anwendungen mit IBM WebSphere 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.

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 bzw. bei z/OS-Systemen mit gemeinsamer Warteschlange den Namen einer Gruppe mit gemeinsamer Warteschlange bereitstellen. 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* können Sie STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING oder ISOLATED_BINDING für alle MQCNO-Versionen festlegen. 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 225.

MQCNO_STANDARD_BINDING

Standardmäßig schließt MQCONNX (wie MQCONN) zwei logische Threads ein, auf denen die WebSphere MQ-Anwendung und der lokale Warteschlangenmanager-Agent in separaten Prozessen ausgeführt werden können. Die WebSphere MQ-Anwendung fordert die WebSphere 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 entweder MQCNO_SHARED_BINDING oder MQCNO_ISOLATED_BINDING, je nach dem Wert des Warteschlangenmanager-Attributs DefaultBindType, das in der Datei qm.ini oder der Windows-Registrierungsdatenbank definiert ist.

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 eine der folgenden Optionen verwendet werden, um das Verhalten von MQCONN oder MQCONNX zu ändern, wenn MQCNO_STANDARD_BINDING angegeben wurde (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 setzen voraus, dass die WebSphere MQ-Anwendung und der lokale Warteschlangenmanager Teil des gleichen Prozesses 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 WebSphere MQ-Threadbibliotheken verknüpfen. Anweisungen zur Definition einer WebSphere 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 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 jedoch Anwendungsprozess und lokaler Warteschlangenmanageragent separat aktiv; sie nutzen keine Ressourcen gemeinsam.

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:

- Unter z/OS wird MQCNO_CLIENT_BINDING mit MQRC_OPTIONS_ERROR abgelehnt.
- 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.
- V 7.5.0.7 In den Versionen vor IBM WebSphere MQ Version 7.5.0Fixpack 7 ist MQCNO_CLIENT_BINDING für .NET nicht verfügbar, da .NET über eigene Verfahren zur Auswahl des BIND-Typs verfügt. Ab Version 7.5.0, Fix Pack 7 besteht diese Einschränkung bezüglich der Verwendung von .NET für MQCNO_LOCAL_BINDING nicht mehr.
- Ist die Umgebungsvariable MQ_CONNECT_TYPE beim Aufruf von MQCONNX nicht gesetzt, wird versucht, unter Verwendung des Standardbindungstyps eine Standardserverbindung 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:

- Unter z/OS wird MQCNO_LOCAL_BINDING 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.
- V 7.5.0.7 In den Versionen vor IBM WebSphere MQ Version 7.5.0Fixpack 7 ist MQCNO_LOCAL_BINDING für .NET nicht verfügbar, da .NET über eigene Verfahren zur Auswahl des BIND-Typs verfügt. Ab Version 7.5.0, Fix Pack 7 besteht diese Einschränkung bezüglich der Verwendung von .NET für MQCNO_LOCAL_BINDING nicht mehr.
- Ist die Umgebungsvariable MQ_CONNECT_TYPE beim Aufruf von MQCONNX nicht gesetzt, wird versucht, unter Verwendung des Standardbindungstyps eine Standardserverbindung herzustellen. Kann die Serverbibliothek nicht geladen werden, wird versucht, eine Clientverbindung herzustellen.

Bei z/OS werden diese Optionen zugelassen, allerdings wird nur eine Standard-Binding-Verbindung ausgeführt. MQCNO Version 3 für z/OS ermöglicht vier alternative 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 WebSphere MQ auf UNIX and Linux-Systemen müssen Sie für alle MQI-Aufrufe 'mqm' als effektive Benutzer- und Gruppen-ID verwenden. Diese IDs können Sie vor einem Nicht-MQI-Aufruf, der eine Authentifizierung erfordert, ändern (z. B. vor dem Öffnen einer Datei), vor dem nächsten MQI-Aufruf *müssen* Sie sie aber wieder auf 'mqm' zurücksetzen.
- Unter WebSphere MQ für HP-UX reicht für Multithread-Direktaufrufanwendungen die Standard-Stackgröße vermutlich nicht aus. 256 KB werden empfohlen.
- Unter WebSphere MQ für Windows werden vertrauenswürdige 64-Bit-Anwendungen nicht unterstützt. Beim Versuch, eine vertrauenswürdige 64-Bit-Anwendung auszuführen, wird diese auf eine Standard-Binding-Verbindung herabgestuft.
- Auf Systemen mit WebSphere MQ unter UNIX and Linux werden keine vertrauenswürdigen 32-Bit-Anwendungen 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: Wird unter WebSphere MQ for z/OS nicht unterstützt.

Auf WebSphere MQ Plattformen (außer auf WebSphere MQ for z/OS) ist eine Verbindung, die mit `MQCONN` hergestellt wurde, nur für den Thread verfügbar, der diese 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.

Außer bei einer MTS-Umgebung (Microsoft Transaction Server) ist der Standardwert MQCNO_HANDLE_SHARE_NONE. In MTS-Umgebungen ist der Standardwert MQCNO_HANDLE_SHARE_BLOCK.

Vom MQCONNX-Aufruf wird eine Verbindungskennung zurückgegeben. Die Kennung kann durch nachfolgende MQI-Aufrufe von jedem Thread im Prozess verwendet werden, der diese Aufrufe der von MQCONNX zurückgegebenen Kennung zuordnet. MQI-Aufrufe, die eine gemeinsam genutzte Einzelkennung verwenden, werden über Threads serialisiert.

Beispielsweise ist bei einer gemeinsam genutzten Kennung folgende Reihenfolge von Vorgängen möglich:

1. Thread 1 gibt MQCONNX aus und ruft eine gemeinsam genutzte Kennung *h1* ab
2. Thread 1 öffnet eine Warteschlange und gibt eine Abrufanforderung mithilfe von *h1* aus
3. Thread 2 gibt eine Einreihungsanforderung mithilfe von *h1* aus
4. Thread 3 gibt eine Einreihungsanforderung mithilfe von *h1* aus
5. Thread 2 gibt MQDISC mithilfe von *h1* aus

Solange die Kennung von einem Thread genutzt wird, steht die Verbindung den anderen Threads nicht zur Verfügung. Sofern es akzeptabel ist, dass ein Thread wartet, bis ein vorangegangener Aufruf eines anderen Threads abgeschlossen ist, können Sie MQCONNX mit der Option MQCNO_HANDLE_SHARE_BLOCK verwenden.

Allerdings können Blockierungen Schwierigkeiten verursachen. Nehmen wir an, dass in Schritt „2" auf Seite 226 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.

Verwendung von MQCONNX-Aufrufoptionen mit MQ_CONNECT_TYPE

In diesem Abschnitt werden die unterschiedlichen MQCONNX-Aufrufoptionen und ihre Verwendung in Verbindung mit MQ_CONNECT_TYPE beschrieben.

Unter WebSphere MQ für IBM i, WebSphere MQ für Windows und WebSphere MQ für UNIX and Linux können Sie die Umgebungsvariable MQ_CONNECT_TYPE zusammen mit dem Bindungstyp verwenden, der im Feld *Options* der in einem MQCONNX-Aufruf verwendeten MQCNO-Struktur angegeben ist.

Tabelle 33. 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	LOKAL	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:

- Unter 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 WebSphere 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.

Wenn Sie MQCONNX zur Verbindung unter WebSphere 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 WebSphere MQ-Objekt zu öffnen oder zu verbinden, das Objekt auch wieder. Selbst wenn einem Job, der eine Verbindung zu einem WebSphere 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 WebSphere MQ-Objekten.

Um eine der folgenden Operationen auszuführen, müssen Sie zuerst das relevante WebSphere 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 248).

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 219.

Es gibt vier Typen von WebSphere 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 WebSphere MQ-Objekten finden Sie im Abschnitt Objekte.

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 250.

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 229
- „Dynamische Warteschlangen erstellen” auf Seite 237
- „Ferne Warteschlangen öffnen” auf Seite 238

- „Objekte mit dem MQCLOSE-Aufruf schließen” auf Seite 239

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

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 221 und „Gemeinsam genutzte (threadunabhängige) Verbindungen mit MQCONNX” auf Seite 225. 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 251 unter „Verteilerlisten" auf Seite 250.

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 WebSphere 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. Einen Überblick über den Vorgang der Namensauflösung finden Sie unter Tabelle 34 auf Seite 231. Der Name, den Sie im Feld *ObjectQMgrName* angeben, wird *vor* dem im Feld *ObjectName* bereitgestellten Namen aufgelöst.

Tabelle 34 auf Seite 231 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 34. Warteschlangennamen bei Verwendung von MQOPEN auflösen

Eingabe in MQOD		**Aufgelöste Namen**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Übertragungswarteschlange**
Leer oder lokaler Warteschlangenmanager	Lokale Warteschlange ohne Clusterattribut	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend (lokale Warteschlange verwendet)
Leerer Warteschlangenmanager	Lokale Warteschlange mit Clusterattribut	Durch Auslastungsmanagement gewählter Cluster-Warteschlangenmanager oder bei PUT-Vorgang gewählter spezifischer Cluster-Warteschlangenmanager	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE und die verwendete lokale Warteschlange SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Lokaler Warteschlangenmanager	Lokale Warteschlange mit Clusterattribut	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend (lokale Warteschlange verwendet)
Leer oder lokaler Warteschlangenmanager	Modellwarteschlange	Lokaler Warteschlangenmanager	Erstellter Name	Nicht zutreffend (lokale Warteschlange verwendet)
Leer oder lokaler Warteschlangenmanager	Aliaswarteschlange mit oder ohne Clusterattribut	Namensauflösung erneut mit unverändertem *ObjectQMgrName* ausführen und *ObjectName* mit Einstellung auf *BaseQName* im Definitionsobjekt der Aliaswarteschlange eingeben. Darf sich nicht in einen lokal definierten Aliasnamen auflösen, in dem *ObjectQMgrName* angegeben ist, kann sich aber in einen Cluster-Aliasnamen auflösen (der auf einem anderen Warteschlangenmanager bereitgestellt wird), in dem *ObjectQMgrName* leer ist.		
Lokaler Warteschlangenmanager	Aliaswarteschlange mit Clusterattribut	Der Aliasname darf nicht in eine Clusterwarteschlange aufgelöst werden, die nicht lokal definiert ist, oder die denselben *ObjectName* wie der Alias hat.		

Tabelle 34. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD		**Aufgelöste Namen**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Übertragungswarteschlange**
Leerer Warteschlangenmanager	Aliaswarteschlange mit Clusterattribut	Der Aliasname kann in eine Clusterwarteschlange mit gleichem *ObjectName* wie der Alias aufgelöst werden.		
Leer oder lokaler Warteschlangenmanager	Lokale Definition einer fernen Warteschlange	Namensauflösung erneut mit *ObjectQMgrName* mit Einstellung auf *RemoteQMgrName* und *ObjectName* mit Einstellung auf *RemoteQName* ausführen. Darf keine fernen Warteschlangen auflösen		Name von *XmitQName*-Attribut, wenn nicht leer; anderenfalls *RemoteQMgrName* in das Definitionsobjekt einer fernen Warteschlange. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Leerer Warteschlangenmanager	Kein übereinstimmendes lokales Objekt; Clusterwarteschlange gefunden	Durch Auslastungsmanagement gewählter Cluster-Warteschlangenmanager oder bei PUT-Vorgang gewählter spezifischer Cluster-Warteschlangenmanager	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Leer oder lokaler Warteschlangenmanager	Kein übereinstimmendes lokales Objekt. Clusterwarteschlange nicht gefunden.		Fehler, Warteschlange nicht gefunden	Nicht zutreffend
Name von Warteschlangenmanager in gleicher Warteschlangengruppe für gemeinsame Nutzung wie lokaler Warteschlangenmanager	Lokale gemeinsam genutzte Warteschlange	Lokaler Warteschlangenmanager	Eingabe *ObjectName*	Nicht zutreffend
Name einer lokalen Übertragungswarteschlange	(Nicht aufgelöst)	Eingabe *ObjectQMgrName*	Eingabe *ObjectName*	Eingabe *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)

Tabelle 34. Warteschlangennamen bei Verwendung von MQOPEN auflösen (Forts.)

Eingabe in MQOD		**Aufgelöste Namen**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Übertragungswarteschlange**
Aliasdefinition für Warteschlangenmanager (*RemoteQMgrName* kann der lokale Warteschlangenmanager sein)	(Nicht aufgelöste, ferne Warteschlange)	Namensauflösung erneut mit *ObjectQMgrName* mit Einstellung *RemoteQMgrName* ausführen. Darf nicht in ferne Warteschlange aufgelöst werden	Eingabe *ObjectName*	Name von *XmitQName*-Attribut, wenn nicht leer; anderenfalls *RemoteQMgrName* in das Definitionsobjekt einer fernen Warteschlange. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Warteschlangenmanager ist nicht der Name eines lokalen Objekts; Cluster-Warteschlangenmanager oder Warteschlangenmanager-Aliasname gefunden	(Nicht aufgelöst)	*ObjectQMgrName* oder bestimmter Cluster-Warteschlangenmanager bei PUT-Vorgang ausgewählt	Eingabe *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)
Warteschlangenmanager ist nicht der Name eines lokalen Objekts; keine Clusterobjekte gefunden	(Nicht aufgelöst)	Eingabe *ObjectQMgrName*	Eingabe *ObjectName*	*DefXmitQName*-Attribut des Warteschlangenmanagers, wo *DefXmitQName* unterstützt wird. SYSTEM.QSG.TRANSMIT.QUEUE (siehe Hinweis)

Anmerkungen:

1. *BaseQName* ist der Name der Basiswarteschlange von der Definition der Aliaswarteschlange.
2. *RemoteQName* ist der Name der fernen Warteschlange von der lokalen Definition der fernen Warteschlange.
3. *RemoteQMgrName* ist der Name des fernen Warteschlangenmanagers von der lokalen Definition der fernen Warteschlange.
4. *XmitQName* ist der Name der Übertragungswarteschlange von der lokalen Definition der fernen Warteschlange.
5. Bei Verwendung von WebSphere MQ for z/OS-Warteschlangenmanagern einer Gruppe mit gemeinsamer Warteschlange kann in Tabelle 34 auf Seite 231 statt des Namens des lokalen Warteschlangenmanagers auch der Name der Gruppe mit gemeinsamer Warteschlange verwendet werden.

   Wenn der lokale Warteschlangenmanager die Zielwarteschlange nicht öffnen oder keine Nachricht darin einreihen kann, wird die Nachricht entweder über die gruppeninterne Steuerung von Warteschlangen oder einen WebSphere MQ-Kanal an den im Parameter 'ObjectQMgrName' angegebenen Warteschlangenmanager übertragen.
6. In der Tabellenspalte *ObjectName* bezieht sich CLUSTER sowohl auf CLUSTER- als auch auf CLUSNL-Attribute der Warteschlange.
7. SYSTEM.QSG.TRANSMIT.QUEUE wird verwendet, wenn lokale und ferne Warteschlangenmanager in der gleichen Warteschlangengruppe mit gemeinsamer Nutzung 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 Übertragungswarteschlange geöffnet. Daher können Sie weder die von Ihnen angegebene Warteschlange löschen, noch die Warteschlange, in die sie auflöst, während die andere offen ist.

Obwohl eine Aliaswarteschlange in eine andere lokal definierte Aliaswarteschlange (egal ob in einem Cluster gemeinsam genutzt oder nicht) nicht aufgelöst werden kann, ist das Auflösen in eine über Fernzugriff definierte Cluster-Aliaswarteschlange zulässig und kann daher als Basiswarteschlange angegeben werden.

Der aufgelöste Name der Warteschlange bzw. des Warteschlangenmanagers werden in der MQOD-Struktur in den Feldern *ResolvedQName* und *ResolvedQMgrName* gespeichert.

Weitere Informationen zur Namensauflösung in einer Umgebung mit verteilter Steuerung von Warteschlangen finden Sie im Abschnitt Was ist die Auflösung von Warteschlangennamen?.

Optionen des MQOPEN-Aufrufs verwenden

Geben Sie im Parameter *Options* des MQOPEN-Aufrufs eine oder mehrere Optionen an, um die Art des Zugriffs zu steuern, der Ihnen auf das Objekt, das Sie öffnen, eingeräumt wird. Mit diesen Optionen können Sie sich folgende Berechtigungen einräumen:

- Warteschlange öffnen und festlegen, dass alle Nachrichten, die in diese Warteschlange eingereiht werden, an die gleiche Instanz dieser Warteschlange übergeben werden
- Warteschlange öffnen, um Nachrichten darin einzureihen
- Warteschlange öffnen, um Nachrichten darin zu durchsuchen
- Warteschlange öffnen, um Nachrichten daraus zu entfernen
- Objekt öffnen, um dieses abzufragen und dessen Attribute festzulegen (Sie können allerdings nur die Attribute von Warteschlangen festlegen)
- Thema oder Themenzeichenfolge öffnen, um Nachrichten darin zu veröffentlichen
- Kontextinformationen mit einer Nachricht verknüpfen
- Alternative Benutzer-ID für Sicherheitsprüfungen angeben
- Aufruf steuern, wenn der Warteschlangenmanager stillgelegt ist

MQOPEN-Option für Clusterwarteschlangen
Die Bindung für die Warteschlangenkennung wird aus dem Warteschlangenattribut *DefBind* übernommen, das den Wert MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP haben kann.

Sollen alle Nachrichten, die mit MQPUT in eine Warteschlange eingereiht werden, über die gleiche Route an den gleichen Warteschlangenmanager geleitet werden, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_ON_OPEN.

Soll das Ziel zum Zeitpunkt der Einreihung mit MQPUT angegeben werden, also auf Nachrichtenbasis, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_NOT_FIXED.

Sollen alle Nachrichten einer Nachrichtengruppe, die mit MQPUT in eine Warteschlange eingereiht werden, der gleichen Zielinstanz zugeordnet werden, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BIND_ON_GROUP.

Bei Verwendung von Nachrichtengruppen mit Clustern muss entweder MQOO_BIND_ON_OPEN oder MQOO_BIND_ON_GROUP angegeben werden, um sicherzustellen, dass alle Nachrichten in der Gruppe an demselben Ziel verarbeitet werden.

Wenn Sie keine dieser Optionen angeben, wird der Standardwert MQOO_BIND_AS_Q_DEF verwendet.

Wenn Sie in der MQOD den Namen eines Warteschlangenmanagers angeben, wird die Warteschlangeninstanz dieses Warteschlangenmanagers ausgewählt. Ist der Warteschlangenmanagername leer, kann jede Instanz ausgewählt werden. Weitere Informationen hierzu finden Sie unter „MQOPEN und Cluster” auf Seite 370.

Wenn Sie mit einer QALIAS-Definition eine Clusterwarteschlange öffnen, werden einige Warteschlangenattribute durch die Aliaswarteschlange und nicht durch die Basiswarteschlange festgelegt. Clusterattribute gehören zu den Attributen der Basiswarteschlangendefinition, die durch die Aliaswarteschlange überschrieben werden. Im folgenden Snippet wird die Clusterwarteschlange beispielsweise mit MQOO_BIND_NOT_FIXED und nicht mit MQOO_BIND_ON_OPENgeöffnet. Die Definition der Clusterwarteschlange steht im gesamten Cluster zur Verfügung, während die Definition der Aliaswarteschlange nur lokal für den Warteschlangenmanager gilt.

```
DEFINE QLOCAL(CLQ1)  CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

MQOPEN-Option zum Einreihen von Nachrichten
Mit der Option MQOO_OUTPUT können Sie eine Warteschlange oder ein Thema für das Einreihen von Nachrichten öffnen.

MQOPEN-Option zum Durchsuchen von Nachrichten
Wenn Sie eine Warteschlange öffnen möchten, um die Nachrichten darin zu *durchsuchen*, verwenden Sie den MQOPEN-Aufruf mit der Option MQOO_BROWSE.

Dadurch wird ein *Anzeigecursor* erstellt, mit dem der Warteschlangenmanager die nächste Nachricht in der Warteschlange ermittelt. Weitere Informationen finden Sie im Abschnitt „Nachrichten in einer Warteschlange durchsuchen (Browsing)” auf Seite 288.

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 250.
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 darausentfernen 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 35 auf Seite 236 wird erläutert, wie sich die Einstellung dieser Attribute auf die Art des Zugriffs auswirkt, die Sie bei Verwendung dieser Option erhalten.

Tabelle 35. 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 35 auf Seite 236 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 250.

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 340).

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 40
Nachrichtenkontext-Informationen ermöglichen der Anwendung, die die Nachricht abruft, Informationen über den Absender der Nachricht zu erhalten.

„Kontextinformationen steuern" auf Seite 246
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.

MQOPEN-Option zum Stilllegen eines Warteschlangenmanagers
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 stillgelegten 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. Der Name des lokalen Warteschlangenmanagers, der die Übertragungswarteschlange hostet, ist *ResolvedQMgrName*.

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 241).

Zum Erstellen einer dynamischen Warteschlange verwenden Sie eine Schablone, eine sogenannte Modellwarteschlange, in Verbindung mit dem Aufruf MQOPEN. Modellwarteschlangen werden mit den

WebSphere MQ-Befehlen oder über die Betriebs- und Steuerkonsolen erstellt. 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.

Weitere Informationen zu dynamischen Warteschlangen finden Sie unter 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 34 auf Seite 231.

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 WebSphere 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 240
- „Nachrichten in eine ferne Warteschlange einreihen” auf Seite 245
- „Nachrichteneigenschaften festlegen” auf Seite 246
- „Kontextinformationen steuern” auf Seite 246
- „Einzelnachricht mit dem MQPUT1-Aufruf in eine Warteschlange einreihen” auf Seite 248
- „Verteilerlisten” auf Seite 250
- „Fälle, in denen der PUT-Aufruf fehlschlägt” auf Seite 255

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

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 241
- „Nachrichten mit der MQMD-Struktur definieren” auf Seite 241
- „Optionen in der MQPMO-Struktur angeben” auf Seite 241
- „Die Daten der Nachricht” auf Seite 244
- „Nachrichten einreihen; Nachrichtenkennungen verwenden” auf Seite 245

Kennungen angeben

Für die Verbindungskennung (*Hconn*) in CICS bei z/OS-Anwendungen können Sie die Konstante MQHC_DEF_HCONN, die den Wert null hat, oder die vom MQCONN- bzw. 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 WebSphere MQ-Nachrichten” auf Seite 10, 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 250). 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 40. Informationen zur Verwendung der MQPMO-Struktur zur Steuerung der Kontextinformationen einer Nachricht finden Sie im Abschnitt „Kontextinformationen steuern” auf Seite 246.

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 250). 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 254.

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 254).

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 253).

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 WebSphere 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 WebSphere 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.

WebSphere MQ fügt Nachrichten unter folgenden Bedingungen Headerinformationen hinzu:

- Wenn Sie eine Nachricht in eine ferne Warteschlange einreihen, fügt WebSphere MQ der Nachricht eine Übertragungsheaderstruktur (MQXQH) hinzu. Diese Struktur enthält den Namen der Zielwarteschlange und den Namen deren Warteschlangenmanagers.
- Wenn WebSphere 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 WebSphere 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 WebSphere 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 *MaxMsg-Length* 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.

Nachrichten einreihen; Nachrichtenkennungen verwenden

In der MQPMO-Struktur stehen zwei Nachrichtenkennungen zur Verfügung: *OriginalMsgHandle* und *New-MsgHandle*. 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 238. Ansonsten unterscheidet sich die Verwendung des Aufrufs MQPUT bzw. MQPUT1 nicht gegenüber einer lokalen Warteschlange.

Weitere Informationen zur Verwendung ferner Warteschlangen und Übertragungswarteschlangen finden Sie im Abschnitt WebSphere MQ Verfahren für verteiltes Messaging .

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 WebSphere MQ Version 7.0 einreihen, die noch keine Nachrichteneigenschaften verarbeiten kann, können Sie im Parameter PropertyControl der Kanaldefinition angeben, wie die Eigenschaften behandelt werden sollen.

Kontextinformationen 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.

Verwenden Sie zum Steuern der Kontextinformationen das Feld *Options* in der MQPMO-Struktur.

Wenn Sie das nicht tun, überschreibt der Warteschlangenmanager Kontextinformationen, die bereits im Nachrichtendeskriptor sind, mit den Identität- und Kontextinformationen, die er für Ihre eigene 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 WebSphere MQ erfolgen, mithilfe einer leeren Benutzer-ID durchgeführt. Einer leeren Benutzer-ID kann keine explizite Berechtigung zu IBM WebSphere MQ-Ressourcen zugewiesen werden; sie wird aber als Element der besonderen Gruppe "nobody" behandelt. Weitere Informationen zur Sondergruppe nobodyfinden Sie unter Referenzinformationen zur Schnittstelle für installierbare Services.

Wenn Sie keine Kontextinformationen benötigen, die ihrer Nachricht zugeordnet werden, verwenden Sie die Option MQPMO_NO_CONTEXT.

In den folgenden Abschnitten dieses Themas wird die Verwendung von Identitätskontext, Benutzerkontext und allem Kontext erklärt.

- „Identitätskontext übergeben” auf Seite 247
- „Benutzerkontext übergeben” auf Seite 247
- „Gesamten Kontext übergeben” auf Seite 247
- „Identitätskontext einstellen” auf Seite 248
- „Benutzerkontext einstellen” auf Seite 248
- „Gesamten Kontext einstellen” auf Seite 248

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 40
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 236
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 230). 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 241).

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

Wird in WebSphere 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 7.5.0.8

Achtung: Verteilerlisten unterstützen nicht die Verwendung von Aliaswarteschlangen, die auf Topic-Objekte verweisen. Ab Version 7.5.0, Fix Pack 8 gibt IBM WebSphere MQ MQRC_ALIAS_BASE_Q_TYPE_ERROR zurück, falls eine Aliaswarteschlange auf ein Topic-Objekt 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 29 auf Seite 251 zeigt die Funktionsweise von Verteilerlisten.

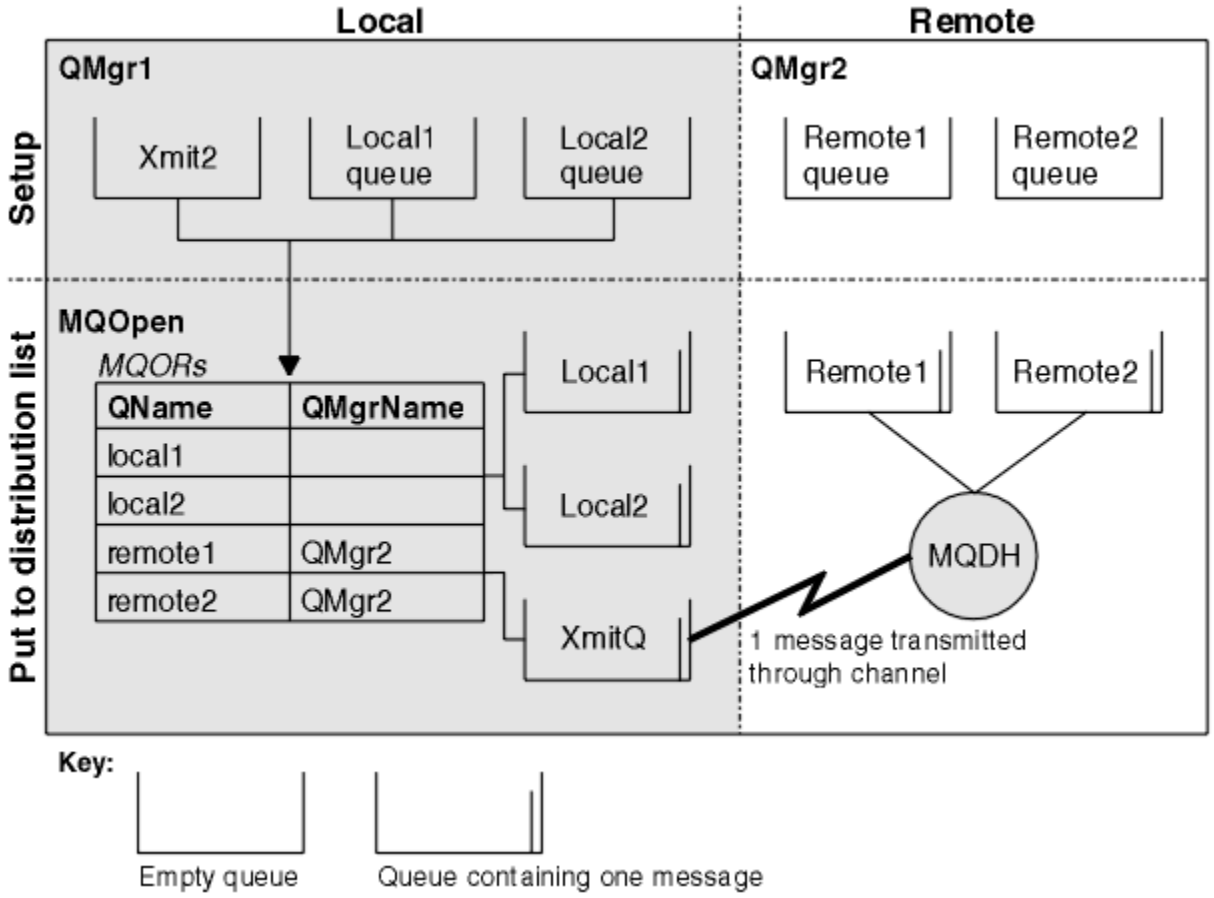


Abbildung 29. So funktionieren Verteilerlisten

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 239)
- 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 beiden, *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 30 auf Seite 253 zeigt, wie Sie eine Verteilerliste in C öffnen.

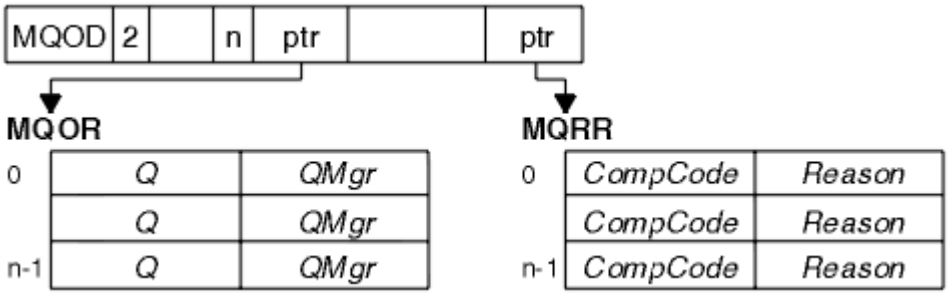


Abbildung 30. Verteilerliste in C öffnen

Abbildung 31 auf Seite 253 zeigt, wie Sie eine Verteilerliste in COBOL öffnen.

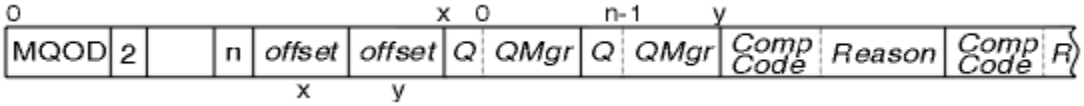


Abbildung 31. Verteilerliste in COBOL öffnen

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 228.

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 239)
- 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 241
- 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 134), 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 32 auf Seite 254 zeigt, wie Sie in C eine Nachricht in eine Verteilerliste einreihen.

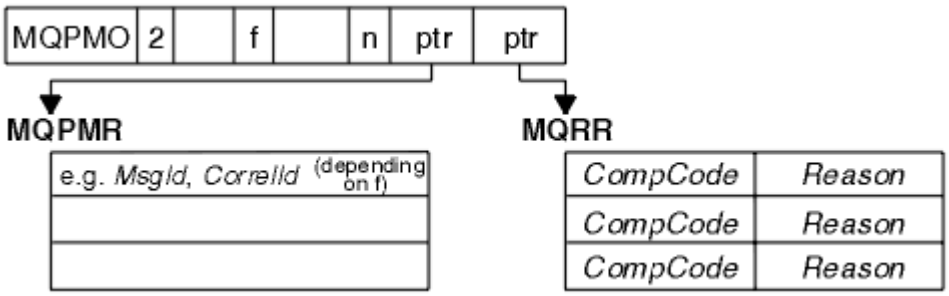


Abbildung 32. Nachricht in C in eine Verteilerliste einreihen

Abbildung 33 auf Seite 254 zeigt, wie Sie in COBOL eine Nachricht in eine Verteilerliste einreihen.

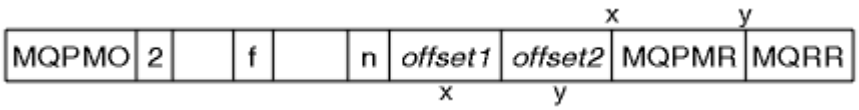


Abbildung 33. Nachricht in COBOL in eine Verteilerliste einreihen

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 219 und „Objekte öffnen und schließen” auf Seite 228.

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 256
- „Abrufreihenfolge der Nachrichten aus einer Warteschlange” auf Seite 261
- „Bestimmte Nachricht abrufen” auf Seite 272
- „Leistung nicht persistenter Nachrichten verbessern” auf Seite 274
- „Nachrichten über 4 MB verarbeiten” auf Seite 278
- „Warten auf Nachrichten” auf Seite 284
-
- „Backout überspringen” auf Seite 285
- „Anwendungsdatenkonvertierung” auf Seite 287
- „Nachrichten in einer Warteschlange durchsuchen (Browsing)” auf Seite 288
- „Fälle, in denen der MQGET-Aufruf fehlschlägt” auf Seite 294

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

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 257
- „Nachrichten mit der MQMD-Struktur und dem MQGET-Aufruf beschreiben" auf Seite 257
- „MQGET-Optionen in der MQGMO-Struktur angeben" auf Seite 258
- „Größe des Pufferbereichs angeben" auf Seite 260

Verbindungskennungen angeben

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 WebSphere MQ-Nachrichten" auf Seite 10, 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 272.

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 261, 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 284)
- 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
- Unter WebSphere MQ for z/OS, ob die abgerufene Nachricht durch 'Backout überspringen' markiert ist (siehe „Backout überspringen” auf Seite 285)
- 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
- Unter WebSphere 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
- Unter WebSphere MQ for z/OS, ob der Aufruf fehlschlägt, falls die Verbindung stillgelegt wurde
- Ob die Konvertierung der Anwendungsnachrichtendaten erforderlich ist (siehe „Anwendungsdatenkonvertierung” auf Seite 287)
- Die Reihenfolge, in der Nachrichten und (mit Ausnahme von WebSphere MQ for z/OS) Segmente aus der Warteschlange abgerufen werden
- Außer unter WebSphere MQ for z/OS, ob nur vollständige logische Nachrichten abgerufen werden können
- Ob die Nachrichten einer Gruppe nur dann abgerufen werden können, wenn *alle* Nachrichten der Gruppe verfügbar sind
- Außer unter WebSphere MQ for z/OS, ob die Segmente einer logischen Nachricht nur dann abgerufen werden können, wenn *alle* Segmente der logischen Nachricht verfügbar sind

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. Unter WebSphere 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 Synchronisationspunkt-steuerung
IBM i	Nein
UNIX and Linux-Systeme	Nein
z/OS	Ja
Windows-Systeme	Nein

- Unter WebSphere MQ for z/OS ist die abgerufene Nachricht nicht für das Überspringen des Backouts 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 eingeht, wird der Aufruf mit einem Ursachencode beendet, der angibt, dass in der Warteschlange keine Nachricht mit den Auswahlkriterien vorhanden war.

Wenn Sie unter WebSphere MQ for z/OS die Option MQGMO_SET_SIGNAL verwenden, legt *WaitInterval* die Zeit fest, nach der das Signal ausgegeben wird.

Weitere Informationen zu diesen Optionen finden Sie in „Warten auf Nachrichten” auf Seite 284 .

Signal1
Signal1 wird nur unter WebSphere MQ for z/OS und MQSeries for HP Integrity NonStop Server unterstützt.

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 WebSphere MQ auf allen anderen Plattformen ist das Feld *Signal1* reserviert und sein Wert ist nicht von Bedeutung.

Signal2
Das Feld *Signal2* ist auf allen Plattformen reserviert und sein Wert ohne Bedeutung.

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 278.

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 18). 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 272.

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.

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 37. 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 34 auf Seite 262 aussehen:

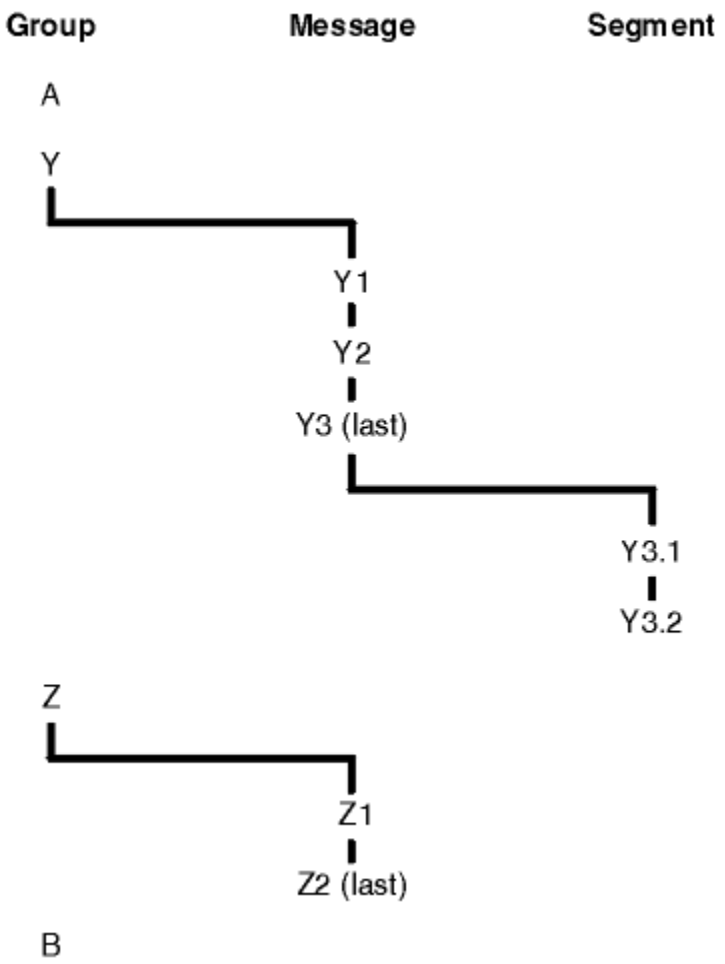


Abbildung 34. Logische Reihenfolge in einer Warteschlange

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 35 auf Seite 263:

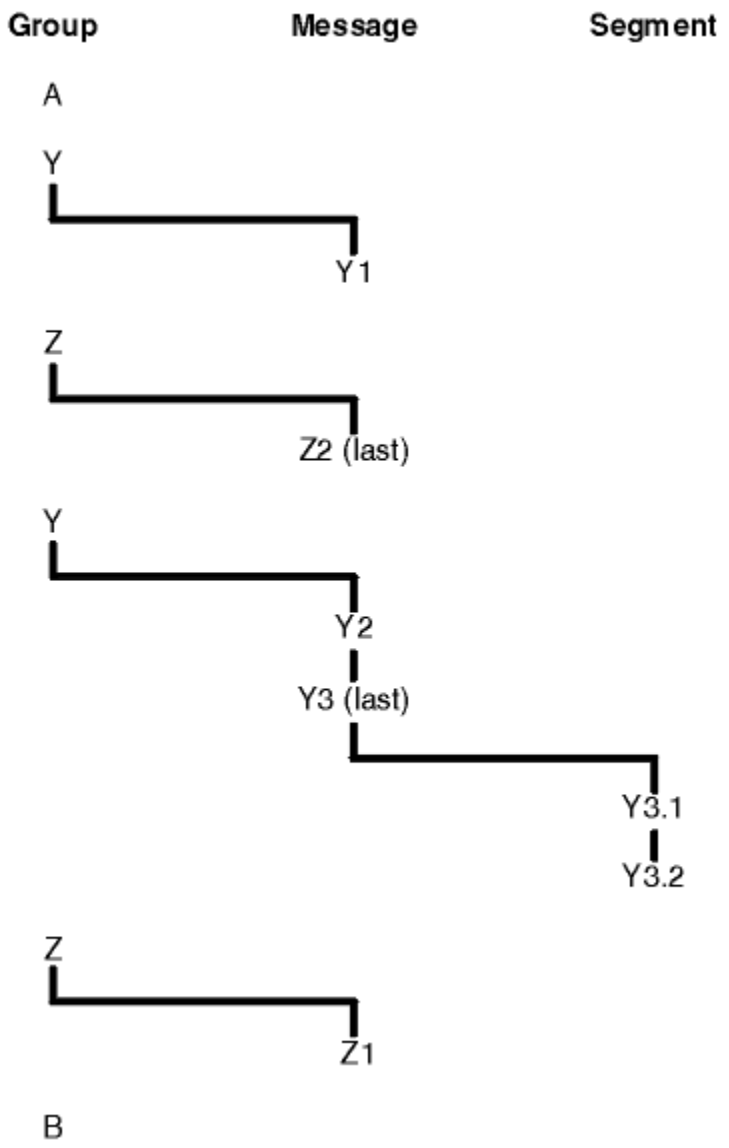


Abbildung 35. Physische Reihenfolge in einer Warteschlange

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 WebSphere 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 an-

geben. Wenn das MQGET mit MQGMO_BROWSE_FIRST umgekehrt nicht MQGMO_LOGICAL_ORDER angibt, müssen die folgenden MQGETs mit MQGMO_BROWSE_NEXT es ebenso wenig angeben.

Die Gruppen- und Segmentinformationen, die der Warteschlangenmanager für MQGET-Aufrufe beibehält, die Nachrichten in der Warteschlange durchsuchen, decken sich nicht mit den Gruppen- und Segmentinformationen, die der Warteschlangenmanager für MQGET-Aufrufe beibehält, die Nachrichten aus der Warteschlangen entfernen. Wenn Sie MQGMO_BROWSE_FIRST angeben, ignoriert der Warteschlangenmanager die Gruppen- und Segmentinformationen für das Durchsuchen und scannt die Warteschlange, als ob es keine aktuelle Gruppe und keine aktuelle logische Nachricht gäbe.

Anmerkung: Verwenden Sie keinen MQGET-Aufruf, um einen Suchvorgang *über das Ende* der Nachrichtengruppe (oder einer logischen Nachricht, die nicht in einer Gruppe ist) hinaus durchzuführen, ohne MQGMO_LOGICAL_ORDER anzugeben. Wenn beispielsweise die letzte Nachricht in der Gruppe *vor* der ersten Nachrichten in der Warteschlangegruppe steht, führt ein Suchvorgang mit MQGMO_BROWSE_NEXT über das Ende der Gruppe hinaus, bei dem MQMO_MATCH_MSG_SEQ_NUMBER angegeben wird und der Parameter *MsgSeqNumber* auf 1 gesetzt ist (um die erste Nachricht in der nächsten Gruppe zu suchen) dazu, das erneut die erste Nachricht in der bereits durchsuchten Gruppe zurückgegeben wird. Dieser Effekt kann sofort eintreten oder mehrere MQGET-Aufrufe später (wenn Zwischengruppen vorhanden sind).

Vermeiden Sie, dass durch *zweifaches* Öffnen der Warteschlange für den Suchvorgang die Möglichkeit einer unendlichen Schleife entsteht:

- Verwenden Sie die erste Kennung, um nur die erste Nachricht in jeder Gruppe anzuzeigen.
- Verwenden Sie die zweite Kennung, um nur die Nachrichten in einer bestimmten Gruppe anzuzeigen.
- Verwenden Sie die MQMO_*-Optionen, um den zweiten Anzeigecursor zur Position des ersten Anzeigecursors zu bewegen, bevor die Nachrichten in der Gruppe durchsucht werden.
- Verwenden Sie MQGMO_BROWSE_NEXT nicht, um einen Suchvorgang über das Ende einer Gruppe hinaus durchzuführen.

Weitere Informationen hierzu finden Sie in den Abschnitten MQGET, MQMD und Regeln zur Überprüfung von MQI-Optionen.

Bei den meisten Anwendungen werden Sie wahrscheinlich entweder eine logische oder physische Reihenfolge für den Suchvorgang wählen. Wenn Sie allerdings zwischen diesen beiden Modi wechseln wollen, beachten Sie, dass Ihre Position innerhalb der logischen Reihenfolge bei der ersten Ausgabe eines Suchvorgangs mit QGMO_LOGICAL_ORDER hergestellt wird.

Wenn das erste Element innerhalb der Gruppe zu diesem Zeitpunkt nicht vorhanden ist, gilt die Gruppe, in der Sie sind, nicht als Teil der logischen Reihenfolge.

Sobald sich der Anzeigecursor innerhalb einer Gruppe befindet, kann er innerhalb derselben Gruppe fortfahren, sogar wenn die erste Nachricht entfernt wird. Zu Beginn können Sie jedoch nie in eine Gruppe mithilfe von MQGMO_LOGICAL_ORDER hineingehen, wenn das erste Element nicht vorhanden ist.

MQPMO_LOGICAL_ORDER

Über die Option MQPMO teilt die Anwendung dem Warteschlangenmanager mit, wie sie Nachrichten in Gruppen und Segmente von logischen Nachrichten einreiht. Sie kann nur für den Aufruf MQPUT angegeben werden; für den Aufruf MQPUT1 ist sie nicht gültig.

Mit MQPMO_LOGICAL_ORDER wird angegeben, dass die Anwendung aufeinanderfolgende MQPUT-Aufrufe für Folgendes verwendet:

1. Segmente werden in jede logische Nachricht nach Systemoffset in aufsteigender Reihenfolge (beginnend bei 0 und ohne Lücken) eingefügt.
2. Alle Segmente werden zunächst vollständig in eine logische Nachricht eingefügt, bevor sie in die nächste logische Nachricht eingefügt werden.
3. Die logischen Nachrichten werden nach Nachrichtenfolgenummer in aufsteigender Reihenfolge (beginnend bei 1 und ohne Lücken) in die einzelnen Nachrichtengruppen eingefügt.IBM WebSphere 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.

In Abbildung 35 auf Seite 263 sind die gültigen Kombinationen von Optionen und Flags sowie die Werte der Felder *GroupId*, *MsgSeqNumber*und *Offset* aufgeführt, die der Warteschlangenmanager jeweils verwendet. 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 36. MQPUT-Optionen zu Nachrichten in Gruppen und Segmenten von logischen Nachrichten.

Angegebene Option				Gruppen- und log-msg-Status vor dem Aufruf		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 bei MQGI_NONE, sonst Feldwert	1	0
Nein	Nein	Ja	Eines	Eines	Eines	Neue Gruppen-ID bei MQGI_NONE, sonst Feldwert	1	Wert in Feld
Nein	Ja	Nein	Eines	Eines	Eines	Neue Gruppen-ID bei MQGI_NONE, sonst Feldwert	Wert in Feld	0
Nein	Ja	Ja	Eines	Eines	Eines	Neue Gruppen-ID bei MQGI_NONE, sonst Feldwert	Wert in Feld	Wert in Feld

Anmerkung:

- MQPMO_LOGICAL_ORDER ist im MQPUT1-Aufruf nicht gültig.
- Falls MQPMO_NEW_MSG_ID oder MQMI_NONE angegeben ist, generiert der Warteschlangenmanager für das Feld *MsgId* eine neue Nachrichten-ID; andernfalls verwendet er den im Feld angegebenen Wert.
- Falls MQPMO_NEW_CORREL_ID angegeben ist, generiert der Warteschlangenmanager für das Feld *CorrelId* eine neue Korrelations-ID; andernfalls verwendet er den im Feld angegebenen Wert.

Bei Angabe von MQPMO_LOGICAL_ORDER setzt der Warteschlangenmanager voraus, dass alle Nachrichten einer Gruppe bzw. alle Segmente einer logischen Nachricht mit dem gleichen Wert im MQMD-Feld *Persistence* eingereiht werden, d. h., alle Nachrichten bzw. Segmente müssen persistent oder alle müssen nicht persistent sein. Ist dies nicht der Fall, schlägt der MQPUT-Aufruf mit dem Ursachencode MQRC_INCONSISTENT_PERSISTENCE fehl.

Die Option MQPMO_LOGICAL_ORDER beeinträchtigt die Arbeitseinheiten wie folgt:

- Wenn die erste physische Nachricht in einer Gruppe oder logischen Nachricht innerhalb einer Arbeitseinheit eingereiht wird, müssen alle anderen physischen Nachrichten in der Gruppe oder logischen Nachricht innerhalb der Arbeitseinheit eingereiht werden, wenn dieselbe Warteschlangenkennung verwendet wird. Sie müssen jedoch nicht innerhalb derselben Arbeitseinheit eingereiht werden, um einer Nachrichtengruppe oder logischen Nachricht, die viele physische Nachrichten beinhaltet, eine Aufteilung über mindestens zwei aufeinanderfolgende Einheiten für die Warteschlangenkennung zu ermöglichen.
- Wenn die erste physische Nachricht in einer Arbeitseinheit oder logischen Nachricht nicht innerhalb einer Arbeitseinheit eingereiht wird, so kann keine der weiteren physischen Nachrichten der Gruppe oder der logischen Nachricht innerhalb einer Arbeitseinheit eingereiht werden, sofern die gleiche Warteschlangenkennung verwendet wird.

Ist dies nicht der Fall, schlägt der MQPUT-Aufruf mit dem Ursachencode MQRC_INCONSISTENT_UOW fehl.

Bei Angabe von MQPMO_LOGICAL_ORDER darf die mit dem MQPUT-Aufruf bereitgestellte MQMD-Version nicht älter als MQMD_VERSION_2 sein. Ist dies nicht der Fall, schlägt der Aufruf mit dem Ursachencode MQRC_WRONG_MD_VERSION fehl.

Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, können Nachrichten in Gruppen und Segmenten von logischen Nachrichten in jeder Reihenfolge eingereiht werden und es ist nicht erforderlich, vollständige Nachrichtengruppen oder vollständige logische Nachrichten einzureihen. Die Anwendung muss sicherstellen, dass die Felder *GroupId*, *MsgSeqNumber*, *Offset* und *MsgFlags* über die entsprechenden Werte verfügen.

Verwenden Sie dieses Verfahren, um eine Nachrichtengruppe oder logische Nachricht zwischendurch erneut zu starten, nachdem ein Systemfehler aufgetreten ist. Beim Neustart des Systems kann die Anwendung die Felder *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags* und *Persistence* auf die korrekten Werte setzen und dann den MQPUT-Aufruf nach Bedarf mit MQPMO_SYNCPOINT oder MQPMO_NO_SYNCPOINT, jedoch ohne Angabe von MQPMO_LOGICAL_ORDER ausgeben. Wenn dieser Aufruf erfolgreich ist, behält der Warteschlangenmanager die Gruppen- und Segmentinformationen bei und nachfolgende MQPUT-Aufrufe können mithilfe dieser Warteschlangenkennung MQPMO_LOGICAL_ORDER wie üblich angeben.

Die vom Warteschlangenmanager für den MQPUT-Aufruf beibehaltenen Gruppen- und Segmentinformationen haben nichts mit den für den MQGET-Aufruf beibehaltenen Informationen zu tun.

Die Anwendung kann für eine bestimmte Warteschlangenkennung MQPUT-Aufrufe, die MQPMO_LOGICAL_ORDER angeben, mit MQPUT-Aufrufen kombinieren, die es nicht angeben. Allerdings sind folgende Punkte zu beachten:

- Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, hat jeder erfolgreiche MQPUT-Aufruf zur Folge, dass der Warteschlangenmanager die Gruppen- und Segmentinformationen für die Warteschlangenkennung auf die von der Anwendung festgelegten Werte einstellt, indem die vorhandenen Gruppen- und Segmentinformationen, die vom Warteschlangenmanager für die Warteschlangenkennung beibehalten werden, ersetzt werden.
- Wenn MQPMO_LOGICAL_ORDER nicht angegeben wird, schlägt der Aufruf nicht fehl, wenn es eine aktuelle Nachrichtengruppe oder logische Nachricht gibt; der Aufruf wird möglicherweise mit einem MQCC_WARNING-Beendigungscode erfolgreich abgeschlossen.Tabelle 37 auf Seite 268 zeigt die verschiedenen Fälle, 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 37. 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. Allerdings benötigen Fachanwendungen unter Umständen mehr Kontrolle als mit der Option MQPMO_LOGICAL_ORDER möglich ist. Das wird umgangen, indem die Option nicht angegeben wird. Wenn Sie sie dennoch angeben, müssen Sie sicherstellen, dass die *GroupId*-, *MsgSeqNumber*-, *Offset*- und *MsgFlags*-Felder in MQMD vor jedem MQPUT- oder MQPUT1-Aufruf ordnungsgemäß eingestellt sind.

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 Datenverkehr 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 37.

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 281 und „Gruppe in einer Arbeitseinheit einreihen und abrufen" auf Seite 270.

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 StatusInfo die *Offset* enthalten.

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. Dieselben Prinzipien gelten, wenn die Segmentierung beteiligt ist (in diesem Fall muss StatusInfo die *Offset* enthalten).

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 38 auf Seite 272 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 38. 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

Tabelle 38. Nachrichten- und Korrelations-IDs verwenden (Forts.)		
Zum Abrufen von	***MsgId***	***CorrelId***
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 39 auf Seite 273 zeigt, welche Nachricht für die möglichen Einstellungen dieser Felder abgerufen wird.

Tabelle 39. 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 261 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 zurücksetzen.

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 23.

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.

Wenn Sie MQOPEN mit MQOO_READ_AHEAD aufrufen, aktiviert der WebSphere MQ-Client Vorauslesen nur unter bestimmten Bedingungen. Zu diesen Bedingungen gehören:

- Client und ferner Warteschlangenmanager müssen beide die WebSphere MQ Version 7 oder höher haben.
- Die Clientanwendung muss kompiliert und mit den WebSphere MQ-Clientbibliotheken mit Thread 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 ist für MQI- und JMS-Anwendungen verfügbar. 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 WebSphere 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 277
Wenn Sie AIX Version 5.3 oder höher verwenden, empfiehlt sich für nicht persistente Nachrichten vermutlich ein Heraufsetzen des Optimierungsparameters auf die volle Leistung.

Zugehörige Tasks

„Vorauslesen aktivieren und inaktivieren" auf Seite 276
Vorauslesen ist standardmäßig inaktiviert. Es kann aber auf Warteschlangen- oder Anwendungsebene aktiviert werden.

Zugehörige Verweise

„MQGET-Optionen und Vorauslesen" auf Seite 275
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.

Wenn Sie MQOPEN mit MQOO_READ_AHEAD aufrufen, aktiviert der WebSphere MQ-Client Vorauslesen nur unter bestimmten Bedingungen. Zu diesen Bedingungen gehören:

- Client und ferner Warteschlangenmanager müssen beide die WebSphere MQ Version 7 oder höher haben.
- Die Clientanwendung muss kompiliert und mit den WebSphere MQ-Clientbibliotheken mit Thread 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 40. MQGET-Optionen und Vorauslesen

	Bei aktiviertem Vorauslesen erlaubt 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	

Tabelle 40. MQGET-Optionen und Vorauslesen (Forts.)			
	Bei aktiviertem Vorauslesen erlaubt 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]**
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

Wenn Sie MQOPEN mit MQOO_READ_AHEAD aufrufen, aktiviert der WebSphere MQ-Client Vorauslesen nur unter bestimmten Bedingungen. Zu diesen Bedingungen gehören:

- Client und ferner Warteschlangenmanager müssen beide die WebSphere MQ Version 7 oder höher haben.
- Die Clientanwendung muss kompiliert und mit den WebSphere MQ-Clientbibliotheken mit Thread verknüpft sein.
- Der Clientkanal muss das TCP/IP-Protokoll verwenden.
- In den Client- und Serverkanaldefinitionen muss für den Kanal ein Wert ungleich null für den Parameter SHARECNV (gemeinsame Dialognutzung) angegeben sein.

So aktivieren Sie Vorauslesen:

- Zum Konfigurieren von Vorauslesen auf Warteschlangenebene setzen Sie das Warteschlangenattribut DEFREADA auf YES.
- Zum Konfigurieren von Vorauslesen auf Anwendungsebene führen sie die folgenden Schritte aus:
  - Wenn Vorauslesen wann immer möglich verwendet werden soll, verwenden Sie den Funktionsaufruf MQOPEN mit der Option MQOO_READ_AHEAD. Wenn allerdings das Warteschlangenattribut DEFREADA auf DISABLED gesetzt ist, kann die Clientanwendung kein Vorauslesen verwenden.
  - Wenn Vorauslesen nur dann verwendet werden soll, wenn diese Funktion für eine Warteschlange aktiviert ist, verwenden Sie den Funktionsaufruf MQOPEN mit der Option MQOO_READ_AHEAD_AS_Q_DEF.

Wenn sich eine Clientanwendung aufgrund ihres Designs nicht für Vorauslesen eignet, können Sie die Funktion wie folgt inaktivieren:

- Auf Warteschlangenebene durch Setzen des Warteschlangenattributs DEFREADA auf NO, wenn Vorauslesen nur auf explizite Anforderung einer Clientanwendung verwendet werden soll, oder auf DISABLED, wenn Vorauslesen unabhängig von einer entsprechenden Anforderung der Clientanwendung auf gar keinen Fall verwendet werden soll.
- Auf Anwendungsebene durch Verwendung des Funktionsaufrufs MQOPEN mit der Option MQOO_NO_READ_AHEAD.

Zwei MQCLOSE-Optionen legen fest, was mit Nachrichten geschieht, die sich beim Schließen der Warteschlange noch im Vorauslesepuffer befinden.

- Verwenden Sie MQCO_IMMEDIATE, um die Nachrichten im Vorauslesepuffer zu verwerfen.
- Verwenden Sie MQCO_QUIESCE, um sicherzustellen, dass die Nachrichten im Vorauslesepuffer vor dem Schließen der Warteschlange von der Anwendung verarbeitet werden. Wenn MQCLOSE mit der Option MQCO_QUIESCE ausgegeben wird und sich noch Nachrichten im Vorauslesepuffer befinden, wird MQRC_READ_AHEAD_MSGS mit MQCC_WARNING zurückgegeben.

Leistung für nicht persistente Nachrichten unter AIX optimieren

Wenn Sie AIX Version 5.3 oder höher verwenden, empfiehlt sich für nicht persistente Nachrichten vermutlich ein Heraufsetzen des Optimierungsparameters auf die volle Leistung.

Wenn der Optimierungsparameter sofort in Kraft treten soll, führen Sie folgenden Befehls als Rootbenutzer aus:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Wenn der Optimierungsparameter sofort in Kraft treten und auch nach einem Neustart noch wirksam sein soll, führen Sie folgenden Befehls als Rootbenutzer aus:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Nicht persistente Nachrichten verbleiben normalerweise nur im Hauptspeicher. Allerdings gibt es Umstände, unter denen AIX nicht persistente Nachrichten zu bestimmten Zeiten auf die Festplatte schreibt. Nachrichten, die auf die Festplatte geschrieben werden, stehen für MQGET so lange nicht zur Verfügung, bis der Speichervorgang auf die Festplatte abgeschlossen ist. Dieser Grenzwert wird durch den vorgeschlagenen Optimierungsbefehl geändert. Anstatt Nachrichten auf die Festplatte zu schreiben, wenn sich

in der Warteschlange 16 KB an Daten angesammelt haben, werden die Nachrichten nach Ausführung dieses Befehls erst auf die Festplatte geschrieben, wenn der Realspeicher des Computers nahezu voll ist. Da es sich hier um eine globale Änderung handelt, können davon auch andere Softwarekomponenten betroffen sein.

Wenn Sie unter AIX Multithread-Anwendungen 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. Dieser Grenzwert gibt den IBM WebSphere MQ -Ursachencode `MQRC_UNEXPECTED_ERROR （2195）` und, falls genügend Dateideskriptoren vorhanden sind, eine IBM WebSphere MQ FFST™ -Datei an.

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
```

Nachrichten über 4 MB verarbeiten

Nachrichten können für die Anwendung, die Warteschlange, oder den Warteschlangenmanager zu groß sein. Je nach Umgebung bietet WebSphere MQ verschiedene Möglichkeiten, Nachrichten zu verarbeiten, die größer als 4 MB sind.

Für das Attribut *MaxMsgLength* kann auf allen WebSphere MQ-Systemen ab V6 ein Wert bis zu 100 MB festgelegt werden. Legen Sie hier einen Wert entsprechend der Größe der Nachrichten der jeweiligen Warteschlange fest. Bei anderen WebSphere MQ-Systemen als WebSphere MQ for z/OS können Sie außerdem:

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 Maximalnachrichtenlänge *standardmäßig* unterstützt wird, hängt von der Umgebung ab.

Diese Attribute können, wenn große Nachrichten verarbeitet werden müssen, auch unabhängig voneinander geändert werden. Der gültige Bereich des Warteschlangenmanagerattributs liegt zwischen 32768 Byte und 100 MB, 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.

Nachrichtensegmentierung

In diesem Abschnitt erfahren Sie, wie Nachrichten segmentiert werden.

Anmerkung: Nicht unterstützt in IBM WebSphere MQ for z/OS oder von Anwendungen, die IBM WebSphere MQ -Klassen für JMS verwenden.

Die Erhöhung der maximalen Nachrichtenlänge, wie im Abschnitt „Maximale Nachrichtenlänge heraufsetzen” auf Seite 278 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 37.

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 270 beschrieben, einreihen und abrufen.

Allerdings können Sie keine segmentierten Nachrichten in globalen Arbeitseinheiten einreihen und abrufen.

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
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.

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
```

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: Wird unter WebSphere 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 WebSphere 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 437.

WebSphere 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 WebSphere 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 *tatsächlich* in den WebSphere 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 WebSphere 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 WebSphere 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 für verteilte Plattformen” auf Seite 101.

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 Siedie 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. (Wenn IMS beendet wird, müssen alle abhängigen Regionen beendet werden.) 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.
- 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.

Weitere Informationen zu den Bedingungen, unter denen diese Aktionen auftreten, finden Sie im Abschnitt MQGMO.

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: Wird nur unter WebSphere 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 WebSphere 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 36 auf Seite 286 zeigt einen typischen Ablauf in einem Anwendungsprogramm, wenn das Backout mit einer MQGET-Anforderung übersprungen werden soll.

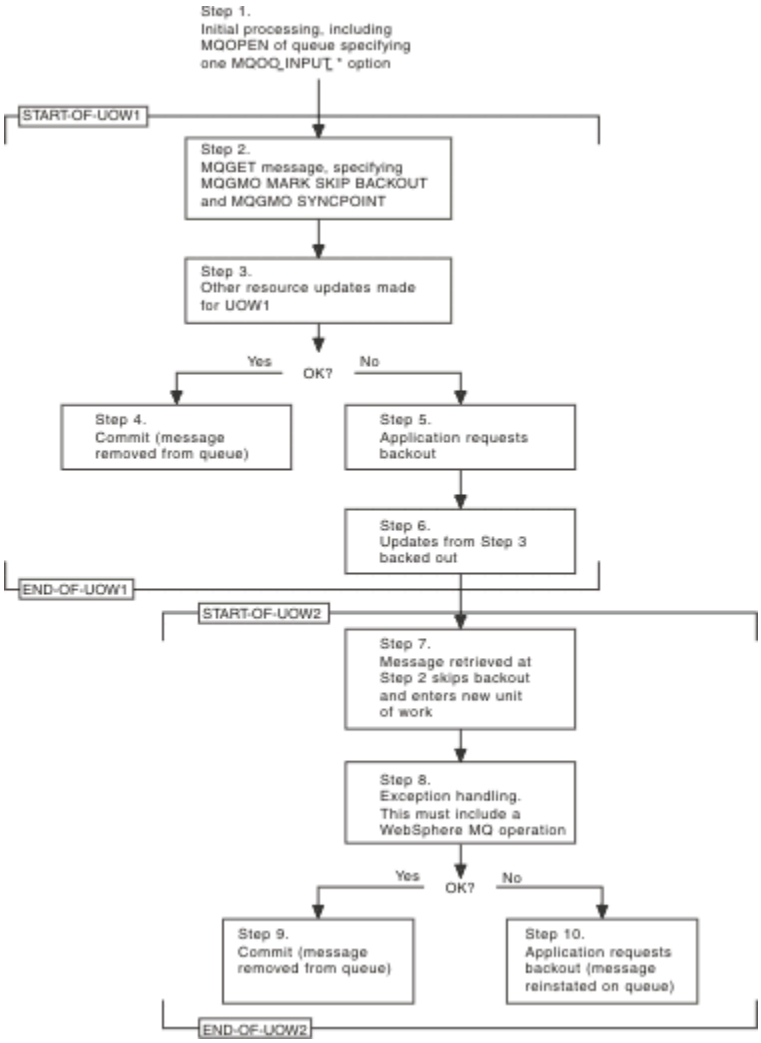


Abbildung 36. Backout mit MQGMO_MARK_SKIP_BACKOUT überspringen

Schritte in Abbildung 36 auf Seite 286:

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 36 auf Seite 286 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 442. 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 WebSphere 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.

   Wenn Sie alle Nachrichten anzeigen möchten, *müssen* Sie die Felder *MsgId* und *CorrelId* der MQMD-Struktur nach jedem MQGET-Aufruf auf null setzen.
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 261.)

- „Warteschlangen in FIFO-Reihenfolge (First In First Out)” auf Seite 289
- „Warteschlangen in Prioritätsreihenfolge” auf Seite 289
- „Nicht festgeschriebene Nachrichten” auf Seite 290
- „Änderung an der Warteschlangenreihenfolge” auf Seite 290
- „Verwendung des Warteschlangenindex” auf Seite 290

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: Wird nur unter WebSphere 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 261 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 291 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_OBJECT_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 WebSphere MQ-Publish/Subscribe-Anwendungen zu entwickeln.

Einen Überblick über Publish/Subscribe-Konzepte finden Sie im Abschnitt Einführung in das Publish/Subscribe-Messaging für WebSphere MQ.

In den folgenden Abschnitten finden Sie Informationen zum Schreiben der verschiedenen Typen von Publish/Subscribe-Anwendungen:

- „Veröffentlichungsanwendungen erstellen” auf Seite 296
- „Subskribentenanwendungen erstellen” auf Seite 303
- „Publish/Subscribe-Lebenszyklen” auf Seite 322
- „Publish/Subscribe-Nachrichteneigenschaften” auf Seite 327
- „Nachrichtenreihenfolge bestimmen” auf Seite 330
- „Veröffentlichungen abfangen” auf Seite 330
- „Veröffentlichungsoptionen” auf Seite 338
- „Subskriptionsoptionen” auf Seite 339

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„IBM WebSphere MQ-Anwendungen entwerfen” auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„WebSphere MQ-Beispielprogramme” auf Seite 100
In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

„Anwendung zur Warteschlangensteuerung schreiben” auf Seite 206
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.

„Clientanwendungen schreiben” auf Seite 374

Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„Web-Services in WebSphere MQ verwenden” auf Seite 1004
Sie können IBM WebSphere MQ-Anwendungen für Web-Services mithilfe des IBM WebSphere MQ-Transports für SOAP oder der IBM WebSphere MQ-Bridge für HTTP entwickeln.

„IBM WebSphere MQ-Anwendung erstellen” auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

„Programmfehler behandeln” auf Seite 581
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.

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 WebSphere MQ -Veröffentlichungsanwendung entspricht dem Schreiben einer WebSphere MQ -Punkt-zu-Punkt-Anwendung, die Nachrichten in eine Warteschlange einreiht (Tabelle 41 auf Seite 296). Der Unterschied besteht darin, dass Sie MQPUT-Nachrichten an ein Thema und nicht an eine Warteschlange senden.

Tabelle 41. Vergleich zwischen Punkt-zu-Punkt- und Publish/Subscribe-WebSphere 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 297), 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 300) sieht das Interaktionsmuster des Programms mit WebSphere 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 303
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine WebSphere MQ-Anwendung, die Nachrichten aus einer Warteschlange konsumiert, eine Anwendung, die eine Subskription erstellt und keine Kenntnisse über die Warteschlan-

gen voraussetzt, und schließlich eine Anwendung, die sowohl Warteschlangen als auch Subskriptionen verwendet.

Zugehörige Verweise
TOPIC DEFINI
ANZEIGEN TOPIC
ANZEIGEN TPSTATUS

Beispiel 1: Veröffentlichungsanwendung für ein festes Thema
Ein WebSphere 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 38 auf Seite 298 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 37. Einfaches WebSphere 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 38. Ausgabe des ersten Beispiels für eine Veröffentlichungsanwendung

Die nachfolgend erläuterten Codezeilen veranschaulichen einzelne Aspekte der Entwicklung einer Veröffentlichungsanwendung für WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ übergeben wird.

Was zeigt dieses erste Beispiel? Das Beispiel ist so nahe wie möglich dem herkömmlichen und bewährten Entwicklungsmuster für WebSphere MQ-Punkt-zu-Punkt-Programme nachempfunden. Ein wichtiger Aspekt bei der Programmierung nach dem WebSphere 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 WebSphere 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 300 veranschaulicht zudem die gemeinsame Verwendung von Themen und Themenzeichenfolgen.

Zugehörige Konzepte

„Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 300

Ein WebSphere MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Programm definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 303
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine WebSphere 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 40 auf Seite 302 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, topicNa
me, topicString);
    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 39. Einfaches WebSphere 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 40. 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 Themenzeichenfolge "NYSE", die vom Themenobjekt STOCKS bereitgestellt wird, mit der vom Programm bereitgestellten Themenzeichenfolge "IBM/PRICE" und fügt einen "/" zwischen den beiden Themenzeichenfolgen ein. Das Ergebnis ist die aufgelöste Themenzeichenfolge "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. WebSphere MQ entnimmt der Themenstruktur in der aufgelösten Themenzeichenfolge implizit, welches administrative Themenobjekt die mit der Veröffentlichung verbundenen Attribute definiert.

Angenommen, es gibt zwei Themenobjekte A und Bund A definiert Thema "a"und B definiert Thema "a/b" (Abbildung 41 auf Seite 303). Wenn das Veröffentlichungsprogramm auf das Themenobjekt A verweist und die Themenzeichenfolge "b"bereitstellt und das Thema in die Themenzeichenfolge "a/b"auflöst, übernimmt die Veröffentlichung die Eigenschaften des Themenobjekts B , da das Thema der für Bdefinierten Themenzeichenfolge "a/b" entspricht.

if (strcmp(argv[1],"/"))
argv[1] ist der optional bereitgestellte Themenname. "/" ist als Themenname ungültig. Hier gibt diese Zeichenfolge an, dass kein Themenname vorliegt und die Themenzeichenfolge vollständig vom Programm bereitgestellt wird. Der Ausgabe in Abbildung 40 auf Seite 302 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 WebSphere 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.

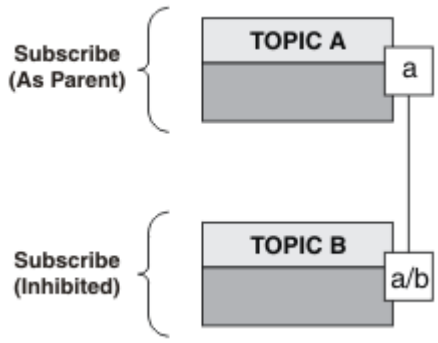


Abbildung 41. Zuordnung der Themenobjekte

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 WebSphere 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 297
Ein WebSphere MQ-Programm zur Veranschaulichung der Veröffentlichung in ein vom Administrator definiertes Thema.

„Subskribentenanwendungen erstellen” auf Seite 303
Sehen Sie sich, bevor Sie selbst Subskribentenanwendungen erstellen, die in dieser Dokumentation bereitgestellten Beispiele an: eine WebSphere 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 WebSphere 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 42 auf Seite 304 mit den für sie typischen WebSphere MQ-Funktionsaufrufsequenzen aufgeführt.

1. Die erste Subskribentenart, MQ-Veröffentlichungskonsument, ist identisch mit einem Punkt-zu-Punkt-MQ-Programm, das nur MQGET durchführt. Die Anwendung hat keinerlei Kenntnisse davon, dass sie Veröffentlichungen konsumiert - sie liest lediglich Nachrichten aus einer Warteschlange. Die Subskription, durch die Veröffentlichungen an die Warteschlange weitergeleitet werden, wird administrativ in WebSphere 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.

Eine Möglichkeit, diese Stile zu verstehen, besteht darin, die C -Beispielprogramme zu studieren, die in Tabelle 42 auf Seite 304 für jeden Stil aufgelistet sind. Die Beispiele wurden passend zu den im Abschnitt „Veröffentlichungsanwendungen erstellen” auf Seite 296 als Beispiele vorgestellten Veröffentlichungsanwendungen entwickelt.

Tabelle 42. Vergleich zwischen Punkt-zu-Punkt- und Subscribe-WebSphere MQ-Programmmustern

Schritt	**MQ-Nachrichtenkonsument**	**„Beispiel 1: Konsument einer MQ-Veröffentlichung” auf Seite 305**	**„Beispiel 2: Verwalteter MQ -Subskribent” auf Seite 307**	**„Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 312**
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 322 beschrieben.

Beispiel 1: Konsument einer MQ-Veröffentlichung

Der MQ-Veröffentlichungskonsument ist ein IBM WebSphere 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 WebSphere 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 312

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 43 auf Seite 306 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, &Re
ason);
        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, subscrip
tionQueue);
        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 42. 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 43. Ausgabe des MQ-Veröffentlichungskonsumenten

Hinsichtlich WebSphere 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 WebSphere 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 zurückgegebene Nachricht eine abschließende Null enthält, falls "`if (messlen == strlen(publication));`" wahr ist. 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 307
Der verwaltete MQ -Subskribent ist das bevorzugte Muster für die meisten Subskribentenanwendungen. Das Beispiel erfordert *keine* Verwaltungsdefinition von Warteschlangen, Themen oder Subskriptionen.

„Beispiel 3: Nicht verwalteter MQ-Subskribent" auf Seite 312
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 296
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 bevorzugte Muster für die meisten Subskribentenanwendungen. Das Beispiel erfordert *keine* Verwaltungsdefinition von Warteschlangen, Themen oder Subskriptionen.

Diese einfachste Art des von verwalteten Subskribenten verwendet normalerweise eine *nicht permanente* Subskription. In diesem Beispiel wird daher auch eine nicht permanente Subskription behandelt. Die Subskription gilt nur so lange wie die Lebensdauer der Subskriptionskennung von `MQSUB`. Alle Veröffentlichungen, die der Themenzeichenfolge während der Lebensdauer der Subskription entsprechen, werden an die Subskriptionswarteschlange gesendet (und möglicherweise an eine ständige Veröffentlichung, wenn das Flag `MQSO_NEW_PUBLICATIONS_ONLY` nicht gesetzt oder standardmäßig festgelegt ist, eine frühere Veröffentlichung, die der Themenzeichenfolge entspricht, beibehalten wurde und die Veröffentlichung persistent war oder der Warteschlangenmanager seit der Erstellung der Veröffentlichung nicht beendet wurde).

Dieses Muster eignet sich auch für *permanente* Subskriptionen. Normalerweise , wenn eine verwaltete permanente Subskription verwendet wird, erfolgt dies aus Gründen der Zuverlässigkeit, anstatt eine Subskription einzurichten, die, ohne dass Fehler auftreten, den Subskribenten überlebt. Weitere Informationen zu verschiedenen Lebenszyklen, die verwalteten, nicht verwalteten, permanenten und nicht permanenten Subskriptionen zugeordnet sind, finden Sie in den zugehörigen Themen.

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. Bei -Bedarfssubskription: Die Topiczeichenfolge für die 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. Selbstverwaltender Subskriptionslebenszyklus: *non-permanente* Subskriptionen sind nur für die Dauer der Subskribentenanwendung vorhanden.
   - Wenn Sie eine *permanente* verwaltete Subskription definieren, führt dies dazu, dass eine permanente Subskriptionswarteschlange und Veröffentlichungen weiterhin in ihr gespeichert werden, ohne dass 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.
   - Ziehen Sie die Einstellung SubExpiry für permanente Subskriptionen in Betracht, damit keine Veröffentlichungen mehr an die Warteschlange gesendet werden und der Subskribent alle verbleibenden Veröffentlichungen konsumieren kann, bevor er die Subskription entfernt und bewirkt, dass der WS-Manager die Warteschlange und alle verbleibenden Veröffentlichungen in ihr löscht.
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 Ausblenden des Stammteils kann eine Anwendung implementiert werden, ohne dass die Anwendung versehentlich eine Themenstruktur erstellt, die sich mit einer anderen Themenstruktur überschneidet, die von einer anderen Instanz oder einer anderen Anwendung erstellt wurde.
5. Verwaltete Themen: Durch Verwendung einer Themenzeichenfolge, in der der erste Teil einem administrativ definierten Themenobjekt entspricht, werden Veröffentlichungen gemäß den Attributen des Themenobjekts verwaltet.
   - Wenn beispielsweise der erste Teil der Themenzeichenfolge mit der Themenzeichenfolge übereinstimmt, die einem Clusterthemenobjekt zugeordnet ist, kann die Subskription Veröffentlichungen von anderen Mitgliedern 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 Attribute für Themen bereit und der Programmierer definiert dynamisch "sub-Themen", ohne sich Gedanken über die Verwaltung von Themen zu machen.
   - Es ist die resultierende Themenzeichenfolge, die verwendet wird, um das Themenobjekt abzugleichen, das die Attribute bereitstellt, die dem Thema zugeordnet sind, und nicht notwendigerweise das Themenobjekt, das in sd.Objectnamebenannt ist, obwohl sie sich normalerweise als ein und dasselbeergeben. Siehe „Beispiel 2: Veröffentlichungsanwendung für ein variables Thema” auf Seite 300.

Wenn die Subskription im Beispiel permanent macht, werden Veröffentlichungen weiterhin an die Subskriptionswarteschlange gesendet, nachdem 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 Subskripti-

onswarteschlange verwendet. Verwaltete Warteschlangen werden mit der Standard-Modellwarteschlange SYSTEM.NDURABLE.MODEL.QUEUE oder SYSTEM.DURABLE.MODEL.QUEUE geöffnet. Sie können die Standardwerte außer Kraft setzen, indem Sie Ihre eigenen permanenten und nicht permanenten Modellwarteschlangen auf Topic-Basis als Eigenschaften des der Subskription zugeordneten 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 verhält sich so, als wäre sie für exklusive Eingabengeöffnet worden.

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 („Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 312) 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 46 auf Seite 311 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 44. Verwalteter MQ -Subskribent-Teil 1: Deklarationen und Parameterbehandlung.

Zu den Deklarationen in diesem Beispiel sind einige Anmerkungen erforderlich:

MQHOBJ Hobj = MQHO_NONE;
Sie können eine nicht permanente verwaltete Subskriptionswarteschlange nicht explizit öffnen, um Veröffentlichungen zu empfangen, aber Sie müssen Speicher für die Objektkennung zuordnen, die der Warteschlangenmanager zurückgibt, wenn er die Warteschlange für Sie öffnet. Es ist wichtig, die Kennung mit MQHO_OBJECTzu initialisieren. Dies gibt dem Warteschlangenmanager an, dass er eine Warteschlangenkennung an die Subskriptionswarteschlange zurückgeben muss.

MQSD sd = {MQSD_DEFAULT};
Der neue, in MQSUB verwendete Subskriptionsdeskriptor.

MQCHAR48 qName;
Obwohl für das Beispiel keine Kenntnisse der Subskriptionswarteschlange erforderlich sind, fragt das Beispiel den Namen der Subskriptionswarteschlange ab-die Bindung MQINQ ist in der Programmiersprache C ein wenig umständlich, sodass Sie diesen Teil des Beispiels für eine Studie nützlich finden können.

```
    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 45. Verwalteter MQ -Subskribent-Teil 2: Codehauptteil.

```
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 46. Ausgabe des verwalteten MQ -Subskribenten

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. Es gibt viele Optionen, in dem Beispiel werden nur die am häufigsten verwendeten Optionen angegeben. Bei den anderen Optionen wird der Standardwert übernommen.

1. Da die Subskription *nicht permanent*ist, d. h., sie eine Lebensdauer der offenen Subskription in der Anwendung hat, setzen Sie das Flag MQSO_CREATE. Sie können auch das Flag *(Standard)* MQSO_NON_DURABLE für die Lesbarkeit festlegen.
2. MQSO_CREATE wird durch MQSO_RESUME ergänzt. Beide Flags können zusammen gesetzt werden. Der Warteschlangenmanager erstellt entweder eine neue Subskription oder nimmt eine vorhandene Subskription wieder auf, je nachdem, was angemessen 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 312 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 WebSphere 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. Diese verhindert, dass das bereitgestellte Nullabschlusszeichen überschrieben wird, falls die aufgelöste Topiczeichenfolge den gesamten Puffer füllt.

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 WebSphere MQ Explorer überprüfen. Sie können den Parameter `sd.SubName` als Eingabeangeben, damit Sie wissen, nach welchem Namen gesucht werden soll. Sie müssen Namenskonflikte mit anderen Subskriptionen vermeiden.

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 305
Der MQ-Veröffentlichungskonsument ist ein IBM WebSphere MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 3: Nicht verwalteter MQ-Subskribent” auf Seite 312
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 296
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. Indem Sie eine permanente Subskription erstellen und die Veröffentlichungsanwendung so einrichten, dass sie permanente Veröffentlichungen erstellt, stellen Sie sicher, dass auch dann keine Nachrichten verloren gehen, wenn der Warteschlangenmanager oder der Subskribent vorzeitig beendet wird und wiederhergestellt werden muss.

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 49 auf Seite 319 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 WebSphere 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 Sie eine vorhandene Subskription in eine andere Warteschlange, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Nehmen Sie eine vorhandene Subskription wieder auf, die einer vorhandenen Warteschlange, `MQSO_RESUME + (Hobj = MQHO_NONE)` oder `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)` zugeordnet ist.

– 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.
– Alternativ gibt der Warteschlangenmanager durch Codierung einer bestimmten Auswahl von MQSO_CREATE | MQSO_RESUME | MQSO_ALTER einen Fehler (Tabelle 43 auf Seite 315) zurück, wenn die Status der Subskription und der Warteschlange, die als Eingabe für MQSUB bereitgestellt werden, nicht mit dem Wert von sd.Optionskonsistent sind. Abbildung 55 auf Seite 322 zeigt die Ergebnisse der Ausgabe von MQSUB für Subskription X mit unterschiedlichen individuellen Einstellungen des Flags sd.Options und der Übergabe von drei verschiedenen Objektkennungen.

Probieren Sie für das in Abbildung 48 auf Seite 318 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 um eine Variante des nicht verwalteten MQ -Subskribenten, aber anstatt MQOPEN und MQSUB in demselben Programm auszugeben, subskribiert ein Programm 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.

Das Verhalten von MQSO_CREATE | MQSO_RESUME | MQSO_ALTER ist wichtig, insbesondere wenn Sie planen, Ihren Code durch die Verwendung von Kombinationen dieser Optionen zu vereinfachen. Sehen Sie sich daher unbedingt die Tabelle Tabelle 43 auf Seite 315 an, die die Ergebnisse zeigt, wenn MQSUB verschiedene Warteschlangenkennungen übergeben werden; sehen Sie sich außerdem die Ergebnisse der Ausführung des Beispielprogramms in Abbildung 50 auf Seite 320 bis Abbildung 55 auf Seite 322 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 43 auf Seite 315wird MQSUB die Subskription X und die Warteschlangenkennung an die Warteschlange Aübergeben. Dies führt zu folgenden Ergebnissen der Subskriptionsoptionen:

- MQSO_CREATE schlägt fehl, da die Warteschlangenkennung der Warteschlange A entspricht, die bereits eine Subskription für Xhat. 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. Im Gegensatz dazu schlägt der Aufruf fehl, wenn die Subskription X nicht in der Warteschlange Avorhanden ist.
- MQSO_ALTER verhält sich in Bezug auf das Öffnen der Subskription und Warteschlange ähnlich wie MQSO_RESUME . Wenn sich die Attribute im Subskriptionsdeskriptor, der an MQSUB übergeben wird, jedoch von den Attributen der Subskription unterscheiden, schlägt MQSO_RESUME fehl, während MQSO_ALTER erfolgreich ist, solange die Programminstanz berechtigt ist, die Attribute zu ändern. Be-

achten Sie, dass Sie die Themenzeichenfolge in einer Subskription nie ändern können. Statt jedoch einen Fehler zurückzugeben, ignoriert MQSUB den Themennamen und die Themenzeichenfolgewerte im Subskriptionsdeskriptor und verwendet die Werte in der vorhandenen Subskription.

Betrachten Sie als Nächstes Tabelle 43 auf Seite 315, in der Subskription X und die Warteschlangenkennung für Warteschlange B an MQSUB übergeben werden. Dies führt zu folgenden Ergebnissen der Subskriptionsoptionen:

- MQSO_CREATE ist erfolgreich und erstellt die Subskription X in Warteschlange B , da dies eine neue Subskription in Warteschlange Bist.
- 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 43. 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
Hobj für *Warteschlange B* an MQSUB übergeben	**MQSO_CREATE** Neue Subskription X wird in Warteschlange B erstellt **MQSO_RESUME** RC = 2019 - Subskriptionswarteschlange entspricht nicht der Warteschlangenobjektkennung **MQSO_ALTER** Subskription X wird aus Warteschlange A in Warteschlange B verschoben	**MQSO_CREATE** Neue Subskription X wird in Warteschlange B erstellt **MQSO_RESUME** RC = 2428 - Subskription X in Warteschlange B nicht vorhanden **MQSO_ALTER** RC = 2428 - Subskription X in Warteschlange B nicht vorhanden

Tabelle 43. MQSUB-Fehler bei verschiedenen Warteschlangenkennungen und Subskriptionskombinationen (Forts.)

	Warteschlange A **Subscription X** **Warteschlange B** **Keine Subskription**	**Warteschlange A** **Keine Subskription** **Warteschlange B** **Keine Subskription**
MQHO_NONE an MQSUB übergeben	**MQSO_CREATE** RC = 2019 - Bad object handle. Stellen Sie das MQSO_MANAGED-Flag so ein, dass eine verwaltete Subskription erstellt wird und erstellen Sie eine verwaltete Warteschlange **MQSO_RESUME** Subskription X in Warteschlange A wird wieder aufgenommen und Hobj wird an Warteschlange A zurückgegeben **MQSO_ALTER** Subskription X in Warteschlange A wird wieder aufgenommen, Hobj wird an Warteschlange A zurückgegeben und erlaubte Änderungen werden vorgenommen	**MQSO_CREATE** RC = 2019 - Bad object handle. Stellen Sie das MQSO_MANAGED-Flag so ein, dass eine verwaltete Subskription erstellt wird und erstellen Sie eine verwaltete Warteschlange **MQSO_RESUME** RC = 2428 - No subscription X **MQSO_ALTER** RC = 2019 - Bad object handle. Keine Warteschlange A oder B

Anmerkung: Der kompakte Codierungsstil wurde aus Gründen der besseren Lesbarkeit verwendet, ist aber nicht für die Übernahme in die Produktion geeignet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIE
SCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

Abbildung 47. 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 48. Nicht verwalteter MQ-Subskribent - Teil 2: Handhabung der Parameter

Zusätzliche Kommentare zur Parameterbehandlung in diesem Beispiel:

switch((argv[5][0]))
Sie haben die Möglichkeit, Alter | C reate | Resume in Parameter 5 einzugeben, um zu testen, wie sich das Überschreiben eines Teils der Optionseinstellung MQSUB , die im Beispiel standardmäßig verwendet wird, auswirkt. Die vom Beispiel verwendete Standardeinstellung ist MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Anmerkung: Die Einstellung MQSO_ALTER oder MQSO_RESUME ohne Einstellung MQSO_DURABLE ist ein Fehler und sd.SubName muss festgelegt werden und auf eine Subskription verweisen, die fortgesetzt 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, entfernt das Beispiel das Flag MQSO_DURABLE . 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 49. Nicht verwalteter MQ-Subskribent - Teil 3: Hauptteil des Codes

Zusätzliche Kommentare zum Code in diesem Beispiel:

if (strlen(subscriptionQueue))
Wenn kein Subskriptionswarteschlangenname vorhanden ist, verwendet das Beispiel MQHO_NONE als Wert für Hobj.

MQOPEN(...);
Die Subskriptionswarteschlange wird geöffnet und die Warteschlangenkennung wird in Hobjgespeichert.

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 die Subskription permanent ist oder nicht, wird die Subskription mit einem impliziten MQCO_KEEP_SUB oder MQCO_REMOVE_SUBgeschlossen. Sie können eine permanente Subskription mit MQCO_REMOVE_SUBschließen, aber eine nicht permanente Subskription mit MQCO_KEEP_SUB *nicht* schließen. Die Aktion von MQCO_REMOVE_SUB besteht darin, die Subskription zu entfernen, wodurch alle weiteren Veröffentlichungen gestoppt werden, die an die Subskriptionswarteschlange gesendet 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 50 auf Seite 320 beginnt das Beispiel mit der Veröffentlichung von 130 im 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 50. Veröffentlichung von 130 unter NYSE/IBM/PRICE

In Abbildung 51 auf Seite 320 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 55 auf Seite 322 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 hängt von der Auswahl oder Kombination aus MQSO_CREATE, MQSO_RESUMEund MQSO_ALTER ab. 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 51. Ständige Veröffentlichung empfangen

In Abbildung 52 auf Seite 321 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 aufge-

nommen. Wenn der Warteschlangennamen angegeben wäre, würde die Warteschlange zuerst geöffnet und die Kennung an MQSUB übergeben werden.

Anmerkung: Der 2038 -Fehler aus MQINQ ist auf die implizite MQOPEN von STOCKTICKER durch MQSUB ohne die Option MQOO_INQUIRE zurückzuführen. 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 52. Subskription wieder aufnehmen

In Abbildung 53 auf Seite 321 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 53. Ständige Veröffentlichung bei neuer, nicht verwalteter, nicht permanenter Subskription empfangen

Zur Veranschaulichung sich überschneidender Subskriptionen wird in Abbildung 54 auf Seite 321 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 54. Überlappende Subskriptionen

Abbildung 55 auf Seite 322 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 Fall verursacht `Alter` den 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 55. Subskriptionsthemen können nicht geändert werden

Zugehörige Konzepte

„Beispiel 1: Konsument einer MQ-Veröffentlichung” auf Seite 305
Der MQ-Veröffentlichungskonsument ist ein IBM WebSphere MQ-Nachrichtenkonsument, der nicht selbst Themen subskribiert.

„Beispiel 2: Verwalteter MQ -Subskribent” auf Seite 307
Der verwaltete MQ -Subskribent ist das bevorzugte Muster für die meisten Subskribentenanwendungen. Das Beispiel erfordert *keine* Verwaltungsdefinition von Warteschlangen, Themen oder Subskriptionen.

„Veröffentlichungsanwendungen erstellen” auf Seite 296
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.

Herkömmliche WebSphere MQ-Objekte wie Warteschlangen werden 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.[2]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 56 auf Seite 324 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.

[2] Natürlich muss auch das Veröffentlichungsprogramm die Passagierdatensätze als persistente Nachrichten senden, um anderen möglichen Ausfällen vorzubeugen.

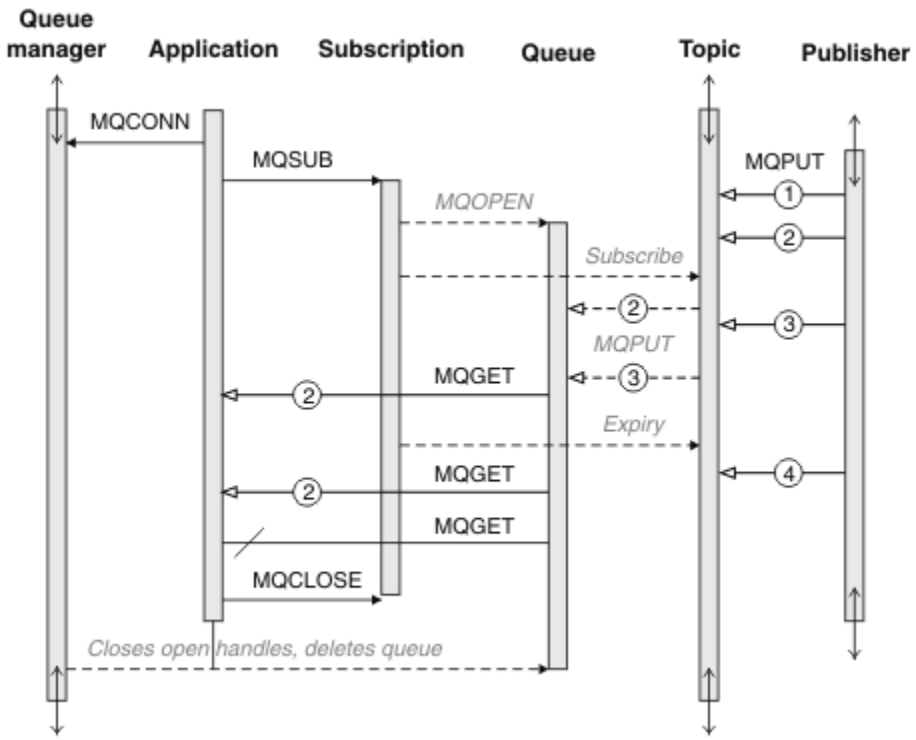


Abbildung 56. Lebenslinien eines verwalteten, nicht permanenten Subskribenten

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 WebSphere MQ-Schnittstelle. Der Warteschlangenmanager kann eine Subskription nur dann zuverlässig verwalten, wenn er die vollständige Kontrolle hat.

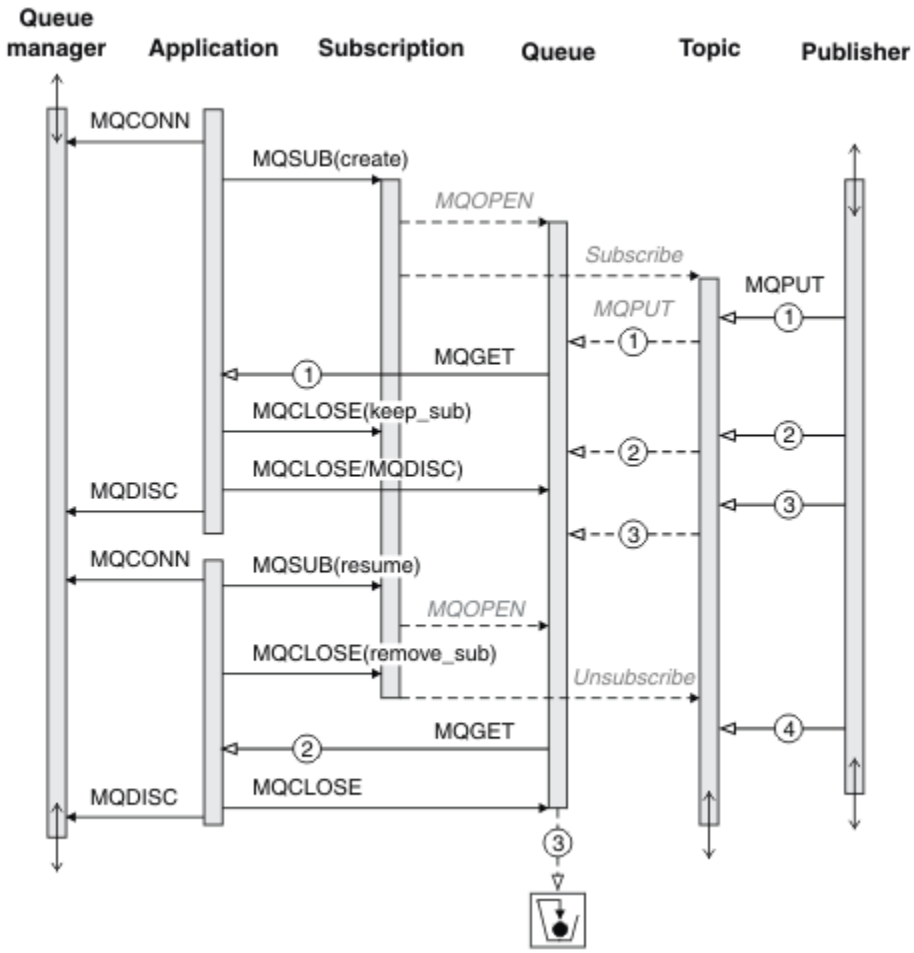


Abbildung 57. Lebenslinien eines verwalteten, permanenten Subskribenten

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.

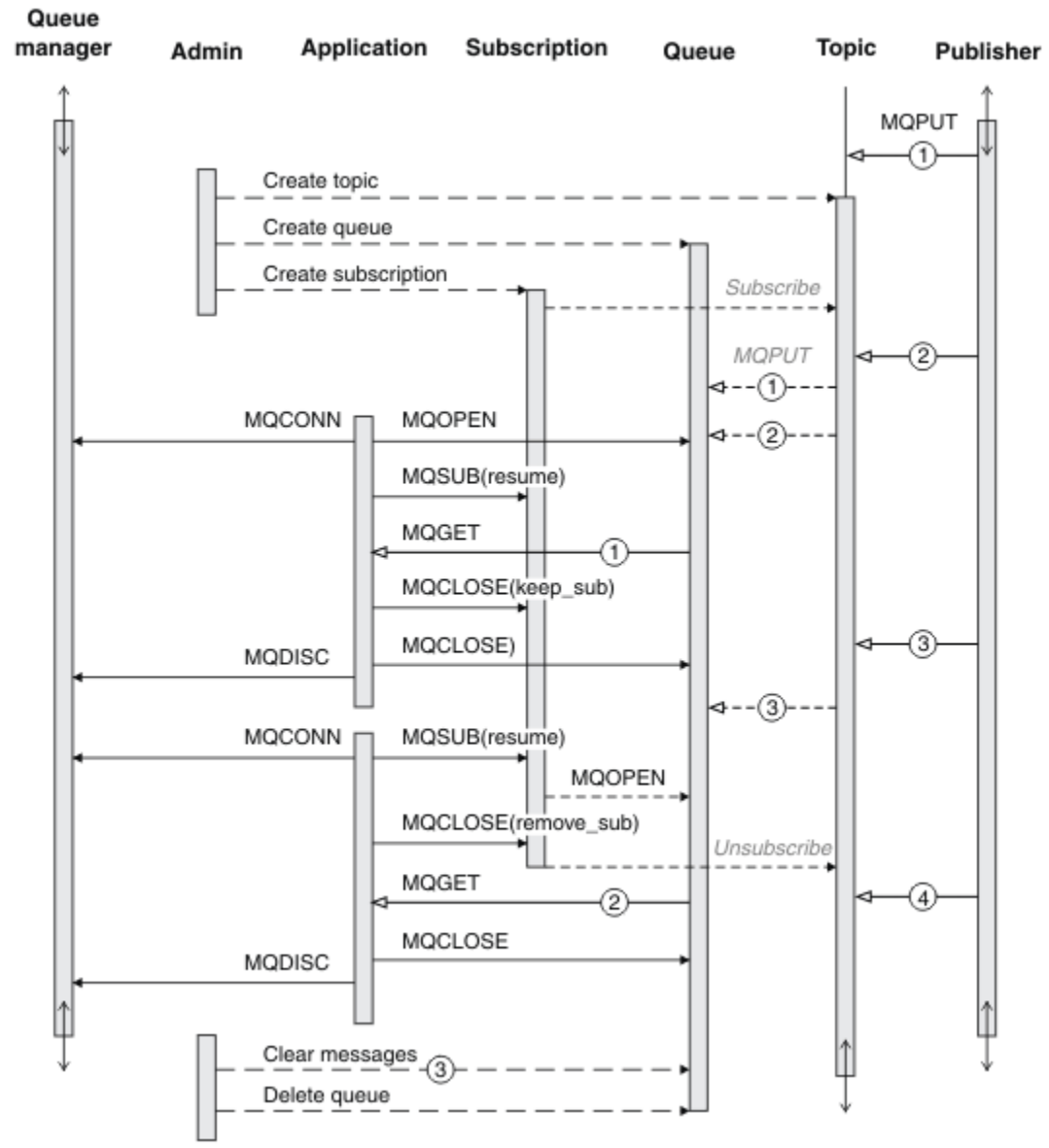


Abbildung 58. Lebenslinien eines nicht verwalteten, permanenten Subskribenten

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

Das Publish/Subscribe-Messaging von WebSphere MQ verwendet verschiedene Nachrichteneigenschaften.

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 40. 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 40. 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 die gleiche Warteschlange verwenden, von der sie ihre Veröffentlichungen erhalten, ermöglicht die Verwendung von MQGET nach Korrelations-ID es, ausschließlich Veröffentlichungen für eine bestimmte Subskription zu erhalten. 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 44 auf Seite 329 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 44. 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 anderen WebSphere MQ-Nachrichten ist es jedoch auch möglich, dass Nachrichten gelegentlich außerhalb dieser Reihenfolge zugestellt werden. Dies kann in folgenden Situationen geschehen:

- Eine Verbindung im Netz wird unterbrochen und nachfolgende Nachrichten werden über eine andere Verbindung umgeleitet.
- Eine Warteschlange ist vorübergehend voll oder gesperrt, so dass eine Nachricht in die Warteschlange für nicht zustellbare Nachrichten eingereiht und deshalb mit Verzögerung zugestellt wird, während nachfolgende Nachrichten auf direktem Wege gesendet werden.
- Ein Administrator löscht einen Warteschlangenmanager, obwohl noch Veröffentlichungsanwendungen und Subskribenten aktiv sind, so dass Nachrichten, die sich noch in Warteschlangen befinden, in die Warteschlange für nicht zustellbare Nachrichten eingereiht und Subskriptionen unterbrochen werden.

Sofern diese Umstände nicht eintreten können, werden Veröffentlichungen immer in der richtigen Reihenfolge zugestellt.

Anmerkung: Gruppierte und segmentierte Nachrichten können mit Publish/Subscribe nicht verwendet werden.

Veröffentlichungen abfangen

Sie können eine Veröffentlichung abfangen, bearbeiten und erneut veröffentlichen, bevor sie einen anderen Subskribenten erreicht.

Eventuell müssen Sie eine Veröffentlichung vor Erreichen eines Subskribenten abfangen, um folgende Aktionen durchzuführen:

- Der Nachricht weitere Informationen hinzufügen
- Die Nachricht blockieren
- Die Nachricht transformieren

Sie können den gleichen Vorgang für jede Nachricht durchführen oder das Vorgehen variieren, abhängig von der Subskription, der Nachricht oder dem Nachrichtenheader.

Zugehörige Verweise

Veröffentlichungsexit - MQ_PUBLISH_EXIT

Subskriptionsebenen

Durch die Subskriptionsebene einer Subskription können Sie eine Veröffentlichung abfangen, bevor sie ihre endgültigen Subskribenten erreicht. Ein abfangender Subskribent subskribiert auf einer höheren Subskriptionsebene und veröffentlicht die Subskription erneut auf einer niedrigeren Subskriptionsebene. Durch eine Kette abfangender Subskribenten können Sie die Nachrichten einer Veröffentlichung verarbeiten, bevor sie dem endgültigen Subskribenten zugestellt wird.

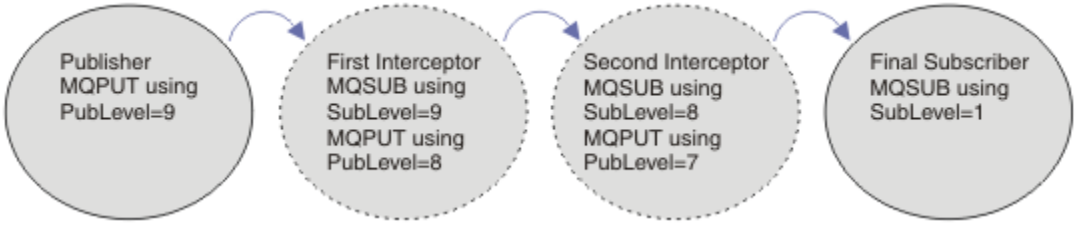


Abbildung 59. Abfolge abfangender Subskribenten

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 45 auf Seite 331 beschriebenen Subskriptionsoptionen.

Tabelle 45. 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.

Tabelle 45. Subskriptionsoptionen für abfangende Subskribenten (Forts.)	
Subskriptionsoption	**Anmerkungen**
PubPriority gleich MQPRI_PRIORITY_AS_PUBLISHED	Die Priorität der abgefangenen Veröffentlichung ändert sich gegenüber der ursprünglichen Veröffentlichung nicht.

Die Optionen in Tabelle 45 auf Seite 331 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 46 auf Seite 332 aufgeführten Nachrichteneigenschaften in die entsprechenden Felder der wiederveröffentlichten Nachricht. Diese Werte können vom abfangenden Subskribenten geändert werden. Mit dem OR-Operator 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 333 an.

Tabelle 46. 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 zum Einreihen von Nachrichten der erneut veröffentlichten Nachricht übertragen (siehe Tabelle 46 auf Seite 332). 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 60 auf Seite 333 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 60. Vorprozessoranweisungen

Abbildung 61 auf Seite 333 listet die in den Codefragmenten verwendeten Deklarationen auf. Mit Ausnahme der hervorgehobenen Stellen handelt es sich hier um die Standarddeklarationen einer WebSphere 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 61. Deklarationen

Initialisierungen, die in den Deklarationen nicht leicht durchführbar sind, werden im Abschnitt Abbildung 62 auf Seite 334 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 WebSphere 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.

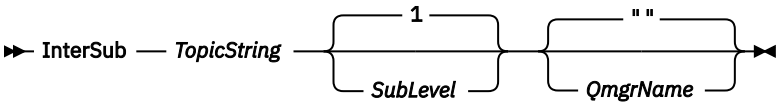
MQPRI_PRIORITY_AS_PUBLISHED
Die Veröffentlichung wird mit ihrer ursprünglichen, bei der Veröffentlichung zugewiesenen Nachrichtenpriorität in die Veröffentlichungswarteschlange eingereiht.

```
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 62. Initialisierungen

Abbildung 63 auf Seite 335 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 63. Abfangen von Veröffentlichungen vorbereiten

Das Hauptcodefragment, Abbildung 64 auf Seite 336 ruft Nachrichten aus der Veröffentlichungswarteschlange ab. Es fragt die Nachrichteneigenschaften ab und veröffentlicht erneut die Nachrichten mit der Topic-Zeichenfolge und den ursprünglichen **MQPMO**.`option`-Eigenschaften der Veröffentlichung.

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 64. Veröffentlichung abfangen und erneut veröffentlichen

Das abschließende Codefragment wird im Abschnitt Abbildung 65 auf Seite 336 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 65. 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 66 auf Seite 337 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.

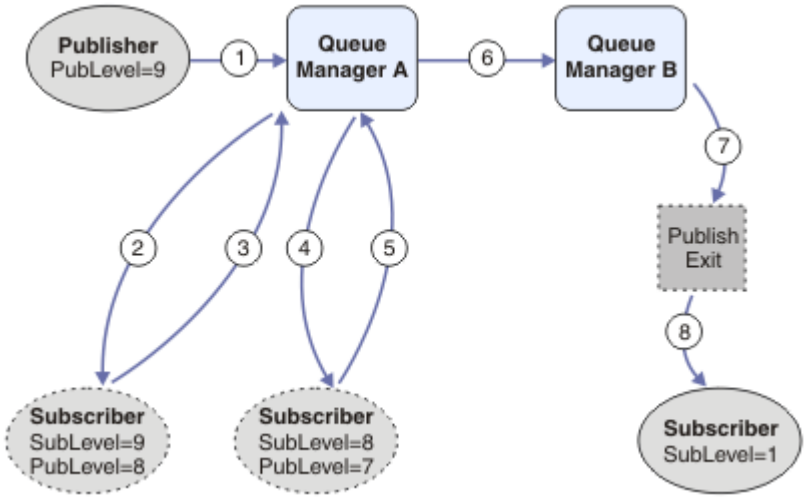


Abbildung 66. Abfangende Subskribenten und Publish-Exit in einem Cluster

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 67 auf Seite 338 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 67 auf Seite 338, dass die Veröffentlichung nicht vom Subskribenten auf `SubLevel` 9 und dem Warteschlangenmanager B abgefangen wird.

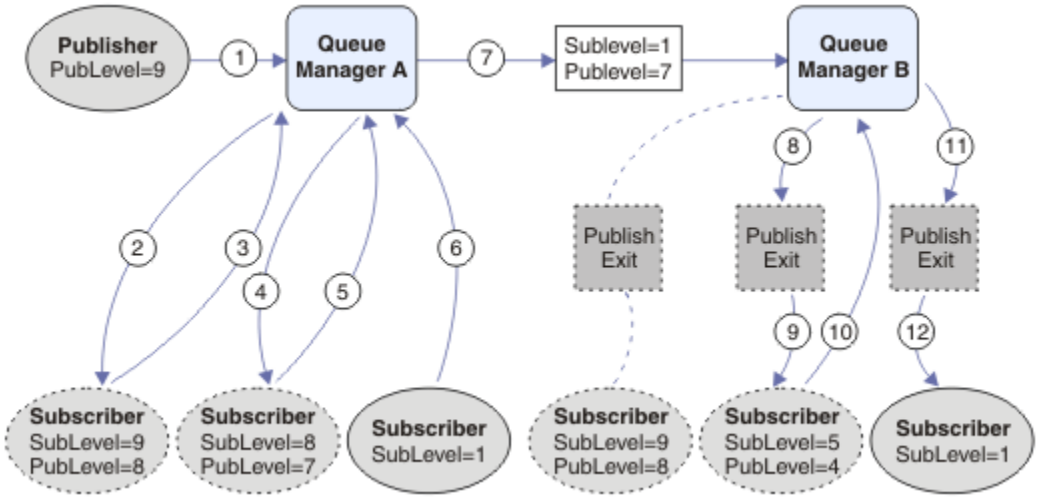


Abbildung 67. Komplexe Implementierung abfangender Subskribenten

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.

Weitere Informationen finden Sie im Abschnitt ../com.ibm.mq.ref.dev.doc/q100080_.dita#q100080_/mqso_group_sub.

Objektattribute abfragen und einstellen

Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

Sie bestimmen, wie ein Warteschlangenmanager ein Objekt verarbeitet. Die Attribute der einzelnen Arten von WebSphere MQ-Objekten werden in Objektattribute näher beschrieben.

Einige Attribute werden bei der Definition eines Objekts festgelegt und können nur mit den WebSphere MQ-Befehlen geändert werden. Ein Beispiel hierfür ist die Standardpriorität für die in Warteschlangen eingereihten Nachrichten. Andere Attribute werden durch die Funktionsweise des Warteschlangenmana-

gers 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 anderer Objekttypen können Sie mit den MQI-Aufrufen nicht ändern. Hierfür müssten Sie folgende Funktionen verwenden:

Für WebSphere MQ für Windows-, UNIX-und Linux -Plattformen

Die MQSC-Funktion (siehe MQSC-Referenz)

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 WebSphere 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 Kennzeichnungder 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:

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 219 und „Objekte öffnen und schließen” auf Seite 228.

Unter den folgenden Links erhalten Sie weitere Informationen zum Abfragen und Einstellen von Objektattributen:

- „Abfragen der Attribute eines Objekts” auf Seite 342
- „Fälle, in denen der MQINQ-Aufruf fehlschlägt” auf Seite 343
- „Warteschlangenattribute einstellen” auf Seite 343

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

Abfragen der Attribute eines Objekts

Mit dem Aufruf MQINQ können Sie die Attribute jedes IBM WebSphere MQ-Objekttyps abfragen.

Als Eingabe für diesen Aufruf müssen Sie Folgendes angeben:

- Eine Verbindungskennung.
- Eine Objektkennung;
- Die Anzahl der Selektoren
- Eine Feldgruppe mit Attributselektoren, wobei jeder Selektor das Format MQCA_* oder MQIA_* aufweisen muss. Jeder Selektor stellt ein Attribut mit einem Wert dar, den Sie abfragen möchten. Außerdem muss jeder Selektor für den Objekttyp gültig sein, den die Objektkennung darstellt. Die Selektoren können in beliebiger Reihenfolge angegeben werden.
- Die Anzahl der Ganzzahlattribute, die Sie abfragen möchten. Geben Sie null an, wenn Sie keine Ganzzahlattribute abfragen.
- Die Länge des Zeichenattributpuffers in *CharAttrLength*. Dies muss mindestens die Summe der Längen sein, die für die einzelnen Zeichenfolgen der Zeichenattribute erforderlich sind. Geben Sie null an, wenn Sie keine Zeichenattribute abfragen.

Die Ausgabe von MQINQ enthält folgende Komponenten:

- Einen Satz mit den Werten der Ganzzahlattribute, kopiert in die Feldgruppe. Die Anzahl der Werte wird von *IntAttrCount* festgelegt. Wenn *IntAttrCount* oder *SelectorCount* null ist, wird dieser Parameter nicht verwendet.
- Den Puffer, in dem Zeichenattribute zurückgegeben werden. Die Länge des Puffers wird durch den Parameter *CharAttrLength* angegeben. Wenn *CharAttrLength* oder *SelectorCount* null ist, wird dieser Parameter nicht verwendet.
- Einen Beendigungscode. Wenn dieser eine Warnung enthält, wurde der Aufruf nur teilweise ausgeführt. Sie sollten in diesem Fall den Ursachencode überprüfen.
- Einen Ursachencode. Es gibt drei Bedingungen, unter denen der Aufruf nur teilweise ausgeführt wird:
  - Der Selektor passt nicht zum Warteschlangentyp
  - Für Ganzzahlattribute ist nicht ausreichend Speicher zugewiesen
  - Für Zeichenattribute ist nicht ausreichend Speicher zugewiesen

  Wenn mehrere dieser Bedingungen zutreffen, wird die zuerst vorgefundene Bedingung zurückgegeben.

Wenn Sie eine Warteschlange für die Ausgabe oder Abfrage öffnen und der Name der Warteschlange löst sich auf eine nicht lokale Clusterwarteschlange auf, können Sie nur den Namen der Warteschlange, den Warteschlangentyp und einige allgemeine Attribute abfragen. Wenn MQOO_BIND_ON_OPEN verwendet wurde, handelt es sich bei den Werten der allgemeinen Attribute um die Werte der ausgewählten Warteschlange. Dagegen handelt es sich bei diesen Werten um die Werte einer beliebigen der möglichen Clusterwarteschlangen, wenn MQOO_BIND_NOT_FIXED oder MQOO_BIND_ON_GROUP verwendet wurde oder wenn MQOO_BIND_AS_Q_DEF verwendet wurde und das Warteschlangenattribut *DefBind* auf MQBND_BIND_NOT_FIXED gesetzt ist. Weitere Informationen hierzu finden Sie in den Abschnitten „MQOPEN und Cluster” auf Seite 370 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 WebSphere 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 auf 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: Sie können das MQI nicht verwenden, um die Attribute von WebSphere MQ-Objekten einzustellen, es sei denn, es handelt sich um lokal definierte Warteschlangen.

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 WebSphere MQ-Anwendungen" auf Seite 345 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 WebSphere 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 WebSphere 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 der Nachrichten innerhalb einer Arbeitseinheit kann auf allen anderen Plattformen außer z/OS mit dem Attribut MAXUMSGS des QMGR-Befehls ALTER gesteuert werden. Einzelheiten zu diesen Befehlen finden Sie in der WebSphere MQ-Scriptbefehlsreferenz (MQSC)*WebSphere MQ Script (MQSC)-Befehlsreferenz*.

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
Eine *zweiphasige Festschreibung* ist ein Prozess, in dem Aktualisierungen, die ein Programm an IBM WebSphere MQ-Warteschlangen vorgenommen hat, mit Aktualisierungen 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 WebSphere 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 Beendi-

gung 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 WebSphere MQ-Anwendungen” auf Seite 345
- „Synchronisationspunkte in IBM WebSphere MQ auf UNIX, Linux, and Windows -Systemen” auf Seite 346

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„IBM WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

„Mit MQI und Clustern arbeiten” auf Seite 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

Überlegungen zu Synchronisationspunkten in IBM WebSphere MQ-Anwendungen

In diesem Abschnitt erhalten Sie Informationen zur Verwendung von Synchronisationspunkten in IBM WebSphere MQ-Anwendungen.

Das zweiphasige Commit wird von folgenden Systemen unterstützt:

- WebSphere MQ for AIX
- WebSphere MQ für HP-UX
- WebSphere MQ für Linux
- WebSphere MQ für Solaris
- WebSphere MQ für Windows
- CICS für MVS/ESA 4.1
- CICS Transaction Server für z/OS
- TXSeries
- IMS/ESA
- Andere externe Koordinatoren, die die X/Open XA-Schnittstelle verwenden

Das einphasige Commit wird von folgenden Systemen unterstützt:

- WebSphere MQ auf UNIX-Systemen
- WebSphere MQ für Windows

Anmerkung: Ausführliche Informationen zu externen Schnittstellen finden Sie im Abschnitt „Schnittstellen zu externen Synchronisationspunktmanagern” auf Seite 349 und in der von der The Open Group herausgegebenen XA-Dokumentation *CAE Specification Distributed Transaction Processing: The XA Specification*. Transaktionsmanager (z. B. CICS, IMS, Encina und Tuxedo) können an zweiphasige Festschreibungen teilnehmen, die mit anderen wiederherstellbaren Ressourcen koordiniert werden. Das bedeutet, dass die Warteschlangenfunktionen, die von WebSphere MQ bereitgestellt werden, innerhalb des Bereichs der Arbeitseinheit verwendet werden und vom Transaktionsmanager verwaltet werden können.

In Beispielen, die mit WebSphere MQ ausgeliefert werden, wird gezeigt, wie WebSphere MQ XA-kompatible Datenbanken koordiniert. Weitere Informationen zu diesen Beispielprogrammen finden Sie im Abschnitt „Beispielprogramme für verteilte Plattformen” auf Seite 101.

In Ihrer WebSphere MQ-Anwendung können Sie bei jedem Put- und Get-Aufruf angeben, ob der Aufruf unter Synchronisationspunktsteuerung erfolgen 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, hängt die Standardaktion von der Plattform ab. Der Standardwert für die Synchronisationspunktsteuerung ist 'no'.

Wenn ein MQPUT1-Aufruf mit MQPMO_SYNCPOINT ausgegeben wird, ändert sich das Standardverhalten, sodass die Put-Operation asynchron beendet wird. Dies kann eine Änderung des Verhaltens einiger Anwendungen verursachen, die bestimmte Felder in den MQOD- und MQMD-Strukturen benötigen, die zurückgegeben werden, aber jetzt undefinierte Werte enthalten. Eine Anwendung kann MQPMO_SYNC_RESPONSE angeben, um sicherzustellen, dass die Put-Operation synchron ausgeführt wird und dass alle zutreffenden Feldwerte abgeschlossen werden.

Wenn Ihre Anwendung auf einen MQPUT- oder MQGET-Aufruf unter Synchronisationspunktsteuerung den Ursachencode MQRC_BACKED_OUT erhält, sollte die Anwendung die aktuelle Transaktion normalerweise mit MQBACK zurücksetzen und anschließend erneut ausführen. Wenn die Anwendung MQRC_BACKED_OUT als Antwort auf einen MQCMIT- oder MQDISC-Aufruf erhält, muss sie MQBACK nicht aufrufen.

Bei jeder Zurücksetzung eines MQGET-Aufrufs wird das Feld *BackoutCount* der MQMD-Struktur der betroffenen Nachricht um eins erhöht. Ein hoher *BackoutCount*-Wert ist ein Hinweis darauf, dass die Nachricht mehrfach zurückgesetzt wurde. Das kann ein Hinweis auf ein Problem mit der Nachricht sein, das Sie untersuchen sollten. Weitere Informationen zu *BackoutCount* finden Sie im Abschnitt BackoutCount.

Wenn ein Programm einen MQDISC-Aufruf ausgibt, solange noch nicht festgeschriebene Anforderungen vorhanden sind, wird ein implizierter Synchronisationspunkt gesetzt. Wenn das Programm fehlerhaft endet, kommt es zu einem impliziten Backout.

Änderungen an den Warteschlangenattributen (entweder durch den MQSET-Aufruf oder durch Befehle) werden nicht durch Festschreiben oder Zurücksetzen von Arbeitseinheiten beeinträchtigt.

Synchronisationspunkte in IBM WebSphere MQ auf UNIX, Linux, and Windows -Systemen

Die Synchronisationspunktunterstützung gilt für lokale und globale Arbeitseinheiten.

Bei einer *lokalen* Arbeitseinheit werden nur die Ressourcen des WebSphere MQ-Warteschlangenmanagers aktualisiert. 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. WebSphere MQ kann selbst solche Arbeitseinheiten koordinieren. Alternativ können sie aber auch durch einen externen Commit-Controller, beispielsweise einen anderen Transaktionsmanager oder den Commitcontroller von IBM i koordiniert werden.

Zur Gewährleistung der Datenintegrität sollten Sie eine zweiphasige Commitprozedur verwenden. Zweiphasige Festschreibung kann von XA-konformen Transaktionsmanagern und Datenbanken wie IBM TXSeries und UDB bereitgestellt werden. WebSphere MQ-Produkte (mit Ausnahme von WebSphere MQ for IBM i und WebSphere MQ for z/OS) können globale Arbeitseinheiten mit einem zweiphasigen Commitprozess koordinieren.

Lokale Arbeitseinheiten

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 WebSphere 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

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 Warteschlangenmanager-Koordination von globalen Arbeitseinheiten wird nicht von WebSphere MQ for IBM i oder WebSphere MQ for z/OS unterstützt. Sie wird in einer WebSphere MQ MQI-Clientumgebung nicht unterstützt.

Hier übernimmt WebSphere 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 steht für die Verbindung zum WebSphere MQ-Warteschlangenmanager.

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. Ein zweiphasiger Festschreibungsprozess wird verwendet, wodurch zuerst alle Ressourcenmanager (z. B. XA-kompatible Datenbankmanager wie DB2, Oracle und Sybase) aufgefordert werden, eine Festschreibung vorzubereiten. 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

Dies tritt ein, wenn ein anderer Synchronisationspunktkoordinator als WebSphere MQ, CICS, Encina oder Tuxedo ausgewählt wurde.

In diesem Fall wird von WebSphere MQ unter UNIX and Linux sowie von WebSphere MQ für Windows beim Synchronisationspunktkoordinator Interesse am Ergebnis der Arbeitseinheit angemeldet, damit alle nicht festgeschriebenen Get- oder Put-Operationen bei Bedarf festgeschrieben oder rückgängig gemacht werden können. Der externe Synchronisationspunktkoordinator legt fest, ob Protokolle für ein ein- oder zweiphasiges Commit bereitgestellt werden.

Wenn Sie einen externen Koordinator verwenden, können MQCMIT, MQBACK und MQBEGIN ausgegeben werden. Aufrufe zu diesen Funktionen schlagen mit dem Ursachencode MQRC_ENVIRONMENT_ERROR fehl.

Die Art, in der eine extern koordinierte Arbeitseinheit gestartet wird, hängt von der Programmierschnittstelle ab, die vom Synchronisationspunktkoordinator bereitgestellt wird. Ein expliziter Aufruf ist möglicherweise erforderlich. Wenn ein expliziter Aufruf erforderlich ist und Sie einen MQPUT-Aufruf ausgeben, der die Option MQPMO_SYNCPOINT angibt, wenn eine Arbeitseinheit nicht gestartet wird, wird der Beendigungscode MQRC_SYNCPOINT_NOT_AVAILABLE zurückgegeben.

Der Umfang der Arbeitseinheit wird durch den Synchronisationspunktkoordinator bestimmt. Der Status der Verbindung zwischen der Anwendung und dem Warteschlangenmanager beeinträchtigt den Erfolg oder Misserfolg von MQI-Aufrufen, die eine Anwendung ausgibt, aber nicht den Status der Arbeitseinheit. Eine Anwendung kann beispielsweise eine Verbindung zu einem Warteschlangenmanager während einer aktiven Arbeitseinheit trennen und wiederherstellen und weitere MQGET- und MQPUT-Operationen innerhalb derselben Arbeitseinheit ausführen. Dies wird als anstehende Verbindungsunterbrechung bezeichnet.

Sie können WebSphere MQ API-Aufrufe in CICS-Programmen verwenden, unabhängig davon, ob sie die XA-Funktionen von CICS verwenden. Wenn Sie XA nicht verwenden, werden die Put- und Get-Operationen zu und von den Warteschlangen nicht innerhalb der atomaren CICS-Arbeitseinheiten 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. Wenn Sie XA verwenden, verwendet CICS ein Protokoll für zweiphasige Festschreibung, um sicherzustellen, dass alle Ressourcen innerhalb der Arbeitseinheit zusammen aktualisiert werden.

Weitere Informationen zur Einstellung von Transaktionsunterstützung finden Sie im Abschnitt „Transaktionsunterstützungsszenarios" auf Seite 43 sowie in der Dokumentation für TXSeries CICS, z. B. *TXSeries für Mehrfachplattformen CICS Administrationsleitfaden für offene Systeme*.

Schnittstellen zu externen Synchronisationspunktmanagern

WebSphere MQ auf UNIX and Linux -Systemen und WebSphere MQ für Windows unterstützen die Koordination von Transaktionen 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 WebSphere MQ auf UNIX-, Linux -und Windows -Systemen hat den Namen MQSeries_XA_RMI. Ausführliche Informationen zu XA-Schnittstellen finden Sie in der von der The Open Group herausgegebenen XA-Dokumentation im Abschnitt *CAE Specification Distributed Transaction Processing: The XA Specification*.

In einer XA-Konfiguration erfüllen WebSphere MQ auf UNIX-, Linux-und Windows -Systemen die Rolle eines XA- Resource Manager. 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:

MQRMIXASwitch	Statisches XA-Ressourcenmanagement
MQRMIXASwitchDynamic	Dynamisches XA-Ressourcenmanagement

Eine Liste der Bibliotheken mit dieser Struktur finden Sie im Abschnitt „XA-Switchstruktur von IBM WebSphere MQ" auf Seite 73.

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 WebSphere MQ mit dem 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 WebSphere 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 225).
2. Auf Windows-Systemen werden alle im XA-Switch deklarierten Funktionen als _cdecl-Funktionen deklariert.
3. 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. Eine JMS-Serveranwendung, die im Bindungsmodus ausgeführt wird, unterliegt immer noch der Regel, dass zur selben Zeit nur jeweils eine Verbindung erlaubt ist.
4. 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. Alternativ können sie bei TXSeries CICS den Exit UE014015 verwenden.

IBM WebSphere MQ-Anwendungen durch Auslöser starten

In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

Einige WebSphere MQ-Anwendungen, die Warteschlangen bedienen, werden unterbrechungsfrei ausgeführt, sodass 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.

WebSphere MQ bietet eine Funktion, mit der eine Anwendung automatisch gestartet werden kann, wenn Nachrichten zum Abrufen bereitstehen. Diese Funktion wird als *Auslösefunktion* bezeichnet.

Informationen zur Kanalauslösung finden Sie im Abschnitt Kanalauslösung.

Das Prinzip der Auslösefunktion

Der Warteschlangenmanager definiert bestimmte Bedingungen als konstituierende *Auslöserereignisse*.

Wenn für eine Warteschlange die Auslösefunktion aktiviert ist und ein Auslöserereignis eintritt, sendet der Warteschlangenmanager eine *Auslösenachricht* an die sogenannte *Initialisierungswarteschlange*. Das Vorhandensein der Auslösenachricht in der Initialisierungswarteschlange zeigt an, dass ein Auslöserereignis aufgetreten ist.

Vom Warteschlangenmanager generierte Auslösenachrichten sind keine persistenten Nachrichten. Dies bedeutet einen geringeren Protokollierungsaufwand (was sich in verbesserter Leistung zeigt) sowie ein Minimum an Duplikaten bei einem Neustart und damit eine verbesserte Neustartzeit.

Die Initialisierungswarteschlange wird von einer sogenannten *Auslösemonitoranwendung* verarbeitet, deren Aufgabe es ist, die Auslösenachricht zu lesen und unter Berücksichtigung der darin enthaltenen Angaben die entsprechende Maßnahme zu ergreifen. In der Regel besteht diese Maßnahme darin, eine andere Anwendung zur Verarbeitung der Warteschlange zu starten, welche die Auslösenachricht generiert hat. Für den Warteschlangenmanager ist die Auslösemonitoranwendung eine ganz gewöhnliche Anwendung, die Nachrichten aus einer Warteschlange (der Initialisierungswarteschlange) liest.

Wenn für eine Warteschlange die Auslösefunktion aktiviert ist, können Sie ein zugehöriges *Prozessdefinitionsobjekt* erstellen. Dieses Objekt enthält Informationen zu der Anwendung, welche die Nachricht verarbeitet, die das Auslöserereignis verursacht hat. Ist das Prozessdefinitionsobjekt erstellt, extrahiert der Warteschlangenmanager diese Informationen und stellt sie in die Auslösenachricht, damit sie von der Auslösemonitoranwendung verwendet werden kann. Der Name der einer Warteschlange zugeordneten Prozessdefinition wird über 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.

Auslösen wird von WebSphere MQ-Clients in folgenden Umgebungen unterstützt:

- UNIX and Linux-Systeme
- Windows-Systeme

Eine Anwendung, die in einer Clientumgebung ausgeführt wird, unterscheidet sich lediglich darin von einer Anwendung in einer vollständigen WebSphere 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 AIX-, HP-UX-, IBM i-, Solaris-, z/OS-oder Windows -Systemen kann 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 (siehe „Bedingungen für ein Auslöserereignis” auf Seite 357). In WebSphere MQ verfügt über eine Reihe von Optionen, mit denen Sie die Bedingungen steuern können, die ein Auslöserereignis verursachen (siehe „Auslöserereignisse steuern” auf Seite 361).

Auslösenachricht
Der Warteschlangenmanager erstellt eine *Auslösenachricht*, sobald er ein Auslöserereignis erkennt (siehe „Bedingungen für ein Auslöserereignis" auf Seite 357). In die Auslösenachricht kopiert er Informationen zu der zu startenden Anwendung. Diese Informationen stammen aus der Anwendungswarteschlange und dem zugehörigen Prozessdefinitionsobjekt. Auslösenachrichten haben ein festgelegtes Format (siehe „Format von Auslösenachrichten" auf Seite 368).

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. Unter WebSphere 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. Unter z/OS wird der Kanalinitiator in der Regel manuell gestartet, der Start kann aber auch automatisch beim Starten eines Warteschlangenmanagers erfolgen, wenn die CSQINP2 in der Start-JCL des Warteschlangenmanagers entsprechend geändert wird. Auf anderen Plattformen wird er automatisch beim Start des Warteschlangenmanagers gestartet oder kann mit dem Befehl 'runmqchi' manuell gestartet werden.

(Weitere Informationen finden Sie im Abschnitt „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore" auf Seite 365.)

Sehen Sie sich zum besseren Verständnis des Auslösemechanismus auch das Beispiel im Abschnitt Abbildung 68 auf Seite 353 an, das den Auslösertyp FIRST (MQTT_FIRST) veranschaulicht.

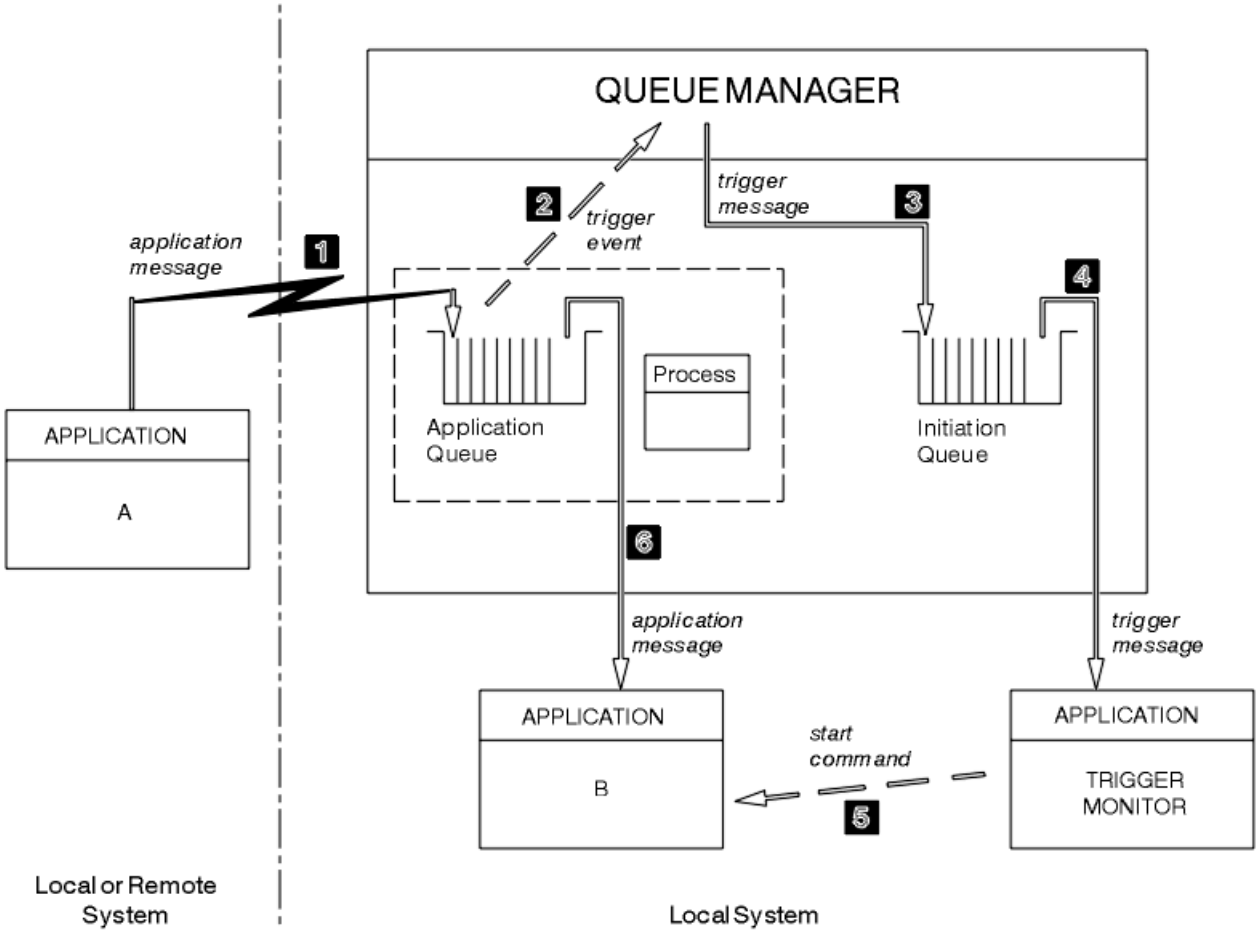


Abbildung 68. Ablauf von Anwendungs- und Auslösenachrichten

In Abbildung 68 auf Seite 353 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 69 auf Seite 354.

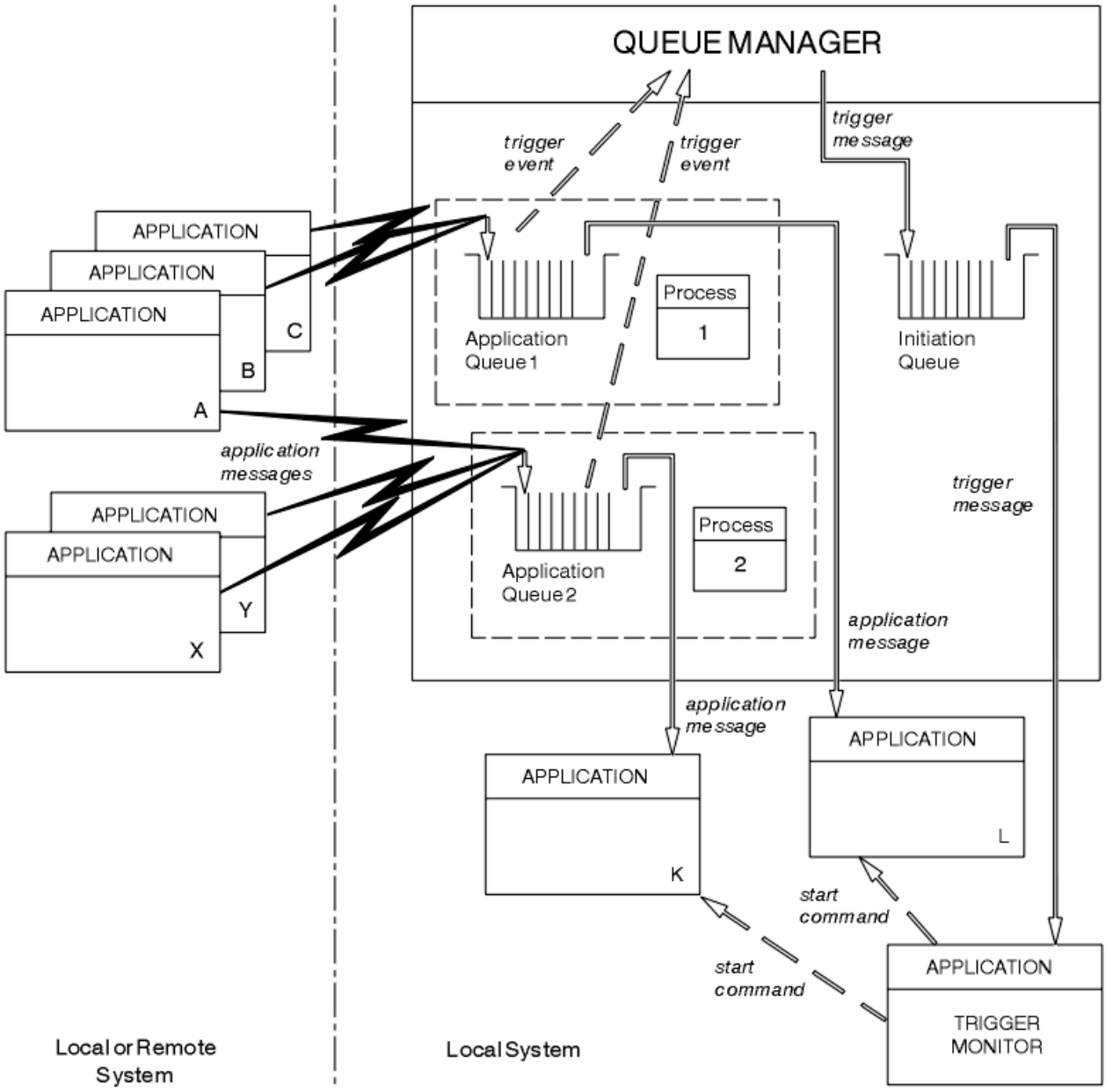


Abbildung 69. Warteschlangenbeziehungen im Zusammenhang mit der Auslösefunktion

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 finden Sie weitere Informationen zum Starten von WebSphere MQ-Anwendungen mithilfe von Auslösern:

- „Voraussetzungen für die Auslösung” auf Seite 355
- „Bedingungen für ein Auslöserereignis” auf Seite 357
- „Auslöserereignisse steuern” auf Seite 361
- „Anwendung entwickeln, die ausgelöste Warteschlangen verwendet” auf Seite 363
- „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore” auf Seite 365
- „Eigenschaften von Auslösenachrichten” auf Seite 367
- „Wenn die Auslösung nicht funktioniert” auf Seite 369

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219
Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 369
In Verbindung mit Clustern gibt es für Aufrufe und Rückkehrcodes spezielle Optionen.

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 wün-

schen 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)
```

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')
   ```

   Wenn der Warteschlangenmanager eine Auslösenachricht erstellt, kopiert er Informationen von den Attributen des Prozessdefinitionsobjekts in die Auslösenachricht.

Plattform	Zum Erstellen eines Prozessdefinitionsobjekts
UNIX-, Linux-und Windows -Systeme	Verwenden Sie DEFINE PROCESS oder verwenden Sie SYSTEM.DEFAULT.PROCESS und ändern Sie es mit ALTER PROCESS

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 WebSphere 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
UNIX-, Linux-und Windows -Systeme	ALTER QLOCAL

6. Starten Sie Instanzen der Auslösemonitore , die die von Ihnen definierten Initialisierungswarteschlangen bedienen sollen. Weitere Informationen finden Sie im Abschnitt „Initialisierungswarteschlangenverarbeitung durch Auslösemonitore” auf Seite 365.

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 361.

Bedingungen für ein Auslöserereignis

Verweise auf gemeinsam genutzte Warteschlangen in diesem Abschnitt beziehen sich auf gemeinsam genutzte Warteschlangen in einer Gruppe mit gemeinsamer Warteschlange, die nur bei WebSphere MQ for z/OS verfügbar sind.

Der Warteschlangenmanager erstellt eine Auslösenachricht, wenn folgende Bedingungen erfüllt werden:

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 ausgelöst.

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 die Anwendungswarteschlange auf WebSphere MQ for z/OS ein *Usage*-Attribut von MQUS_NORMAL hat, werden GET-Anforderungen dafür nicht unterdrückt (das bedeutet, dass das Warteschlangenattribut *InhibitGet* MQQA_GET_ALLOWED heißt). 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 für die Initialisierungswarteschlange werden 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 357 bis „10" auf Seite 358 (mit Ausnahme von „3" auf Seite 357) erfüllt sind, wenn im Fall von MQTT_FIRST seit der Ausgabe der letzten Auslösenachricht für diese Warteschlange ein ausreichender Zeitraum (Warteschlangenmanagerattribut *TriggerInterval*) verstrichen 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* für das Auslöseintervall, *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 357), und die Bedingungen „6" auf Seite 358 bis „10" auf Seite 358 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 39.
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 357), und die Bedingungen „4" auf Seite 357 bis „10" auf Seite 358 (mit Ausnahme von „8" auf Seite 358) 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 357), und die Bedingungen „4" auf Seite 357 bis „10" auf Seite 358 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 357), und die Bedingungen „4" auf Seite 357 bis „10" auf Seite 358, mit Ausnahme von „5" auf Seite 358, 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 357), und die Bedingungen „4" auf Seite 357 bis „10" auf Seite 358 (mit Ausnahme von „8" auf Seite 358) 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 361.

Wenn MSGDLVSQ=PRIORITY festgelegt ist, werden Nachrichten nur dann zu einem Auslöserereignis *hochgezählt*, wenn die Nachrichtenpriorität größer oder gleich der *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 359 (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 WebSphere MQ eine Nachricht während MQPUT segmentiert, wird ein Auslöserereignis erst verarbeitet, wenn alle Segmente erfolgreich in der Warteschlange eingegliedert worden sind. Sobald sich jedoch Nachrichtensegmente in der Warteschlange befinden, behandelt WebSphere MQ sie als einzelne Nachrichten für Auslösezwecke. 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 sorgt jedoch dafür, dass ihre eigenen Auslöserereignisse verarbeitet werden, wenn sie sich durch das WebSphere MQ-Netz bewegen.

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 Priorität unter *TriggerMsgPriority* in der Anwendungswarteschlange eintrifft, ignoriert der Warteschlangenmanager die Nachricht, wenn er bestimmt, ob eine Auslösenachricht erstellt werden soll. Wenn *TriggerMsgPriority* auf null gesetzt ist, werden alle Nachrichten auf ein Auslöserereignis angerechnet.

TriggerType
Zusätzlich zum Auslösertyp NONE (der die Auslösung ebenso inaktiviert wie die Einstellung *TriggerControl* auf OFF) können Sie die folgenden Auslösertypen verwenden, um die Sensitivität einer Warteschlange für Auslöserereignisse festzulegen:

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. Siehe auch „Sonderfall des Auslösertyps FIRST” auf Seite 363.

TIEFE	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:** Bei der Auslösung nach Tiefe inaktiviert der Warteschlangenmanager die Auslösung (mit dem Attribut < xph> < pv>TriggerControl< /pv> < /xph>), nachdem er eine Auslösenachricht erstellt hat. 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 357 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 im Warteschlangenmanagerattribut *TriggerInterval*definiert. Der Standardwert sind 999 999 999 Millisekunden.
- Unter WebSphere MQ for z/OS werden Anwendungswarteschlangen, die sich auf eine offene Initialisierungswarteschlange beziehen, regelmäßig gescannt. Wenn *TRIGINT* Millisekunden vergangen sind, seit die letzte Auslösenachricht gesendet wurde, und 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 Sie unter WebSphere MQ für z/OS *TRIGINT* auf einen niedrigen Wert setzen und es keine Anwendung gibt, die die Anwendungswarteschlange des Auslösertyps FIRST bedient, generiert die Backstop-Auslösung bei jedem regelmäßigen Scan von Anwendungswarteschlangen, die offene Initialisierungswarteschlangen benennen, eine Auslösenachricht.
- Wenn eine Arbeitseinheit zurückgesetzt wird (siehe Auslösenachrichten und Arbeitseinheiten) und das Auslöseintervall hoch bzw. auf den Standardwert eingestellt ist, wird beim Zurücksetzen der Arbeitseinheit eine Auslösenachricht erstellt. Wenn Sie das Auslöseintervall jedoch auf einen niedrigen Wert oder auf null (d. h. Auslösertyp FIRST verhält sich wie Auslösertyp EVERY) gesetzt haben, können viele Auslösenachrichten generiert werden. Wird die Arbeitseinheit zurückgesetzt, werden trotzdem alle Auslösenachrichten zur Verfügung gestellt. Wie viele Auslösenachrichten generiert werden, ist vom Auslöseintervall abhängig. Ist das Auslöseintervall auf 0 gesetzt, wird die maximale Anzahl Nachrichten generiert.

Anwendung entwickeln, die ausgelöste Warteschlangen verwendet

Sie haben gesehen, wie das Einrichten, Steuern und Auslösen bei Ihren Anwendungen funktioniert. Nun folgen ein paar Tipps, was Sie bei der Entwicklung Ihrer Anwendung beachten sollten.

Auslösenachrichten und Arbeitseinheiten

Auslösenachrichten, die durch Auslöserereignisse erstellt werden, die kein Teil einer Arbeitseinheit sind, werden in eine Initialisierungswarteschlange außerhalb der Arbeitseinheiten eingereiht, unabhängig von anderen Nachrichten, und sind für den Abruf durch den Auslösemonitor sofort verfügbar.

Auslösenachrichten, die durch Auslöserereignisse erstellt werden, die Teil einer Arbeitseinheit sind, werden in der Initialisierungswarteschlange zur Verfügung gestellt, wenn die Arbeitseinheit aufgelöst wird, egal ob die Arbeitseinheit festgeschrieben oder zurückgesetzt wird.

Wenn der Versuch des Warteschlangenmanagers, eine Auslösenachricht in eine Initialisierungswarteschlange einzureihen, fehlschlägt, wird sie in eine Warteschlange für nicht zustellbare Nachrichten eingereiht.

Anmerkung:

1. Der Warteschlangenmanager zählt sowohl festgeschriebene als auch nicht festgeschriebene Nachrichten, wenn er beurteilt, ob die Bedingungen für ein Auslöserereignis erfüllt werden.

   Durch das Auslösen von Typ FIRST oder DEPTH werden Auslösenachrichten zur Verfügung gestellt, sogar wenn die Arbeitseinheit zurückgesetzt wird, sodass eine Auslösenachricht immer verfügbar ist, wenn die erforderlichen Bedingungen erfüllt werden. Beispiel: Angenommen eine Put-Anforderung innerhalb eine Arbeitseinheit wird mit Auslösertyp FIRST ausgelöst. Dadurch wird der 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.

   Das bedeutet jedoch, dass Auslösenachrichten manchmal erstellt werden, wenn die Bedingungen für ein Auslöserereignis *nicht* erfüllt werden. Anwendungen, die das Auslösen verwenden, müssen auf diese Situation vorbereitet sein, um damit umgehen zu können. Es wird empfohlen, den MQGET-Aufruf mit Angabe der Option für Warten zu verwenden und im Feld *WaitInterval* einen geeigneten Wert festzulegen.

   Erstellte Auslösenachrichten werden stets zur Verfügung gestellt, egal ob die Arbeitseinheit zurückgesetzt oder festgeschrieben wird.
2. Bei lokalen gemeinsam genutzten Warteschlangen (das heißt, gemeinsam genutzte Warteschlangen in einer Gruppe mit gemeinsamer Warteschlange) zählt der Warteschlangenmanager nur festgeschriebene Nachrichten.

Abrufen von Nachrichten aus einer ausgelösten Warteschlange

Bedenken Sie beim Entwickeln von Anwendungen, die mit Auslösevorgängen arbeiten, dass zwischen dem Start durch einen Auslösemonitor und der Verfügbarkeit der Nachrichten in einer Anwendungswarteschlange eine Verzögerung auftreten kann. Das kann passieren, wenn die Nachricht, die das Auslöserereignis veranlasst, vor den anderen festgeschrieben wird.

Verwenden Sie stets die Option für Warten, wenn Sie den MQGET-Aufruf zum Löschen von Nachrichten aus einer Warteschlange verwenden, für die die Auslöserbedingungen eingestellt sind, um den Nachrichten Zeit bis zum Eintreffen einzuräumen. Der Wert *WaitInterval* sollte so dimensioniert sein, dass er für die längste in angemessenem Rahmen anzunehmende Zeit zwischen der Einreihung einer Nachricht und der Festschreibung dieses Put-Aufrufs ausreicht. Wenn die Nachricht von einem fernen Warteschlangenmanager eintrifft, wird diese Zeit durch Folgendes beeinträchtigt:

- Die Anzahl der Nachrichten, die vor der Festschreibung eingereiht werden
- Die Geschwindigkeit und Verfügbarkeit der Kommunikationsverbindung
- Die Größe der Nachrichten

Beachten Sie dasselbe Beispiel, das wir für die Beschreibung der Arbeitseinheit verwendeten, ebenfalls für eine Situation, in der Sie den MQGET-Aufruf mit der Option für Warten verwenden. Dies war eine Put-Anforderung innerhalb einer Arbeitseinheit für eine Warteschlange, die mit dem Auslösertyp FIRST ausgelöst wird. Durch dieses Ereignis wird der Warteschlangenmanager veranlasst, eine Auslösenachricht zu erstellen. Wenn eine weitere Put-Anforderung von einer anderen Arbeitseinheit erfolgt, wird dadurch kein weiteres Auslöserereignis veranlasst, weil sich die Anzahl der Nachrichten in der Anwendungswarteschlange nicht von Null auf Eins geändert hat. Wird nun die erste Arbeitseinheit zurückgesetzt, die zweite

hingegen festgeschrieben, wird dennoch eine Auslösenachricht erstellt. Daher wird die Auslösenachricht in dem Moment erstellt, in dem die erste Arbeitseinheit zurückgesetzt wird. Wenn es eine bedeutende Verzögerung gibt, bevor die zweite Nachricht festgeschrieben wird, muss die ausgelöste Anwendung möglicherweise darauf warten.

Beim Auslösen des Typs DEPTH kann eine Verzögerung auftreten, sogar wenn alle relevanten Nachrichten schließlich festgeschrieben werden. Angenommen, das Warteschlangenattribut *TriggerDepth* hat den Wert 2. Wenn in diesem Fall zwei Nachrichten in der Warteschlange eintreffen, bewirkt die zweite Nachricht die Generierung einer Auslösenachricht. Wenn jedoch die zweite Nachricht die erste Nachricht ist, die festgeschrieben werden soll, ist das der Moment, in der die Auslösenachricht wieder verfügbar ist. Der Auslösemonitor startet das Serverprogramm, aber das Programm kann die zweite Nachricht nur abrufen, bis die erste festgeschrieben wird. Daher muss das Programm möglicherweise warten, bis die erste Nachricht zur Verfügung gestellt wird.

Entwickeln Sie Ihre Anwendung so, dass sie beendet wird, wenn keine Nachrichten zum Abruf verfügbar sind, wenn Ihr Warteintervall abläuft. Wenn eine oder mehrere Nachrichten später eintreffen, können Sie sich darauf verlassen, dass auf Ihre Anwendung erneut ausgelöst wird, um sie zu verarbeiten. Durch dieses Verfahren wird eine Inaktivität der Anwendung und eine unnötige Verwendung von Ressourcen vermieden.

Initialisierungswarteschlangenverarbeitung durch Auslösemonitore

Für einen Warteschlangenmanager ist ein Auslösemonitor eine Anwendung, die wie jede andere Anwendung eine Warteschlange bedient. Ein Auslösemonitor bedient jedoch Initialisierungswarteschlangen.

Ein Auslösemonitor ist normalerweise ein kontinuierlich ausgeführtes Programm. Wenn eine Auslösenachricht eine Initialisierungswarteschlange erreicht, erhält der Auslösemonitor diese Nachricht. Er verwendet Informationen in der Nachricht, um einen Befehl zum Starten einer Anwendung abzusetzen, die die Nachrichten in der Anwendungswarteschlange verarbeiten soll.

Der Auslösemonitor muss dem Programm, das die Anwendung startet, ausreichende Informationen übergeben, damit das Programm auf der Anwendungswarteschlange die richtigen Aktionen ausführen kann.

Ein Kanalinitiator ist ein Beispiel für einen Sondertyp des Auslösemonitors für Nachrichtenkanalagenten. In dieser Situation müssen Sie jedoch entweder den Auslösertyp FIRST oder den Auslösertyp DEPTH verwenden.

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 für verteilte Plattformen” auf Seite 101.

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 WebSphere MQ die Anzahl der ausführbaren Anwendungen hingegen nicht ein.

Zur Ausführung der gestarteten Anwendung im Hintergrund auf Windows-Systemen geben Sie im Feld *ApplId* als Präfix zum Anwendungsnamen einen START-Befehl ein. Beispiel:

```
START ?B AMQSECHA
```

Um die gestartete Anwendung im Hintergrund auf UNIX -Systemen auszuführen, fügen Sie ein & am Ende der *EnvData* der PROCESS-Definition ein.

Anmerkung: 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 WebSphere MQ-Client bereitgestellt:

runmqtmc
Dieser Monitor ist identisch zu 'runmqtrm', mit der Ausnahme, dass er eine Verknüpfung zu den WebSphere MQ-Clientbibliotheken herstellt.

Für CICS
Der Auslösemonitor 'amqltmc0' wird für CICS bereitgestellt. Er funktioniert genauso wie der Standardauslösemonitor runmqtrm, jedoch starten Sie ihn auf eine andere Weise, und er löst CICS-Transaktionen aus.

Dieser Abschnitt gilt nur für Windows-, UNIX- und Linux-Systeme.

Der Auslösemonitor wird als CICS-Programm bereitgestellt. Sie müssen ihn mit einem vierstelligen Transaktionsnamen definieren. Zum Starten geben Sie einen vierstelligen Namen ein. Es verwendet den Standardwarteschlangenmanager (wie in der Datei qm.ini oder unter WebSphere MQ für Windowsin der Registry angegeben) und das SYSTEM.CICS.INITIATION.QUEUE.

Wenn Sie einen anderen Warteschlangenmanager bzw. eine andere Warteschlange verwenden möchten, erstellen Sie die Auslösemonitorstruktur MQTMC2. Dazu müssen Sie ein Programm mit dem Aufruf 'EXEC

CICS START' schreiben, da die Struktur zu lang ist, um sie als Parameter hinzuzufügen. Diese MQTMC2-Struktur übergeben Sie dann als Daten an die START-Anforderung für den Auslösemonitor.

Bei Verwendung der MQTMC2-Struktur müssen Sie dem Auslösemonitor nur die Parameter *StrucId*, *Version*, *QName* und *QMgrName* übergeben, da er keine anderen Felder referenziert.

Die Nachrichten werden mit 'EXEC CICS START' aus der Initialisierungswarteschlange eingelesen und zum Starten von CICS-Transaktionen verwendet (vorausgesetzt, APPL_TYPE in der Auslösenachricht ist 'MQAT_CICS'. Das Einlesen der Nachrichten aus der Initialisierungswarteschlange erfolgt unter der Synchronisationspunktsteuerung von CICS.

Beim Starten und Stoppen des Monitors sowie bei Fehlern werden Nachrichten generiert. Diese Nachrichten werden an die CSMT-Warteschlange mit transienten Daten gesendet.

Der Auslösemonitor ist in den folgenden Versionen verfügbar:

Version	Verwenden Sie
amqltmc0	TXSeries für AIX, HP-UX und Sun Solaris Version 5.1
amqltmc4	TXSeries für Windows, Version 5.1
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.
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.

Weitere Informationen zur Auslösemonitorschnittstelle (TMI) finden Sie im Abschnitt MQTMC2.

Eigenschaften von Auslösenachrichten

Im folgenden Abschnitt werden weitere Eigenschaften von Auslösenachrichten beschrieben.

- „Persistenz und Priorität von Auslösenachrichten” auf Seite 367
- „Neustart des Warteschlangenmanagers und Auslösenachrichten” auf Seite 368
- „Auslösenachrichten und Änderungen bei Objektattributen” auf Seite 368
- „Format von Auslösenachrichten” auf Seite 368

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. Auf allen WebSphere MQ-Produkten außer auf WebSphere MQ for z/OS kann dieses Feld verwendet werden, um den Namen des auszulösenden Kanals anzugeben.

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. Wenn Sie Auslösemonitor CKTI

oder CSQQTRMN verwenden, die in WebSphere MQ for z/OS bereitgestellt werden, ist das Attribut *ApplId* des Prozessdefinitionsobjekts eine CICS- oder 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. Verwenden Sie nicht die für WebSphere MQ for z/OS bereitgestellten Auslösemonitore (CKTI oder CSQQTRMN) in diesem Feld. Stattdessen können Sie andere Auslösemonitore dafür wählen.

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 Tracedienstprogramm 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 370
- „MQPUT, MQPUT1 und Cluster” auf Seite 371
- „MQINQ und Cluster” auf Seite 372
- „MQSET und Cluster” auf Seite 373
- „Rückgabecodes” auf Seite 373

Zugehörige Konzepte

„Message Queue Interface (MQI) - Übersicht” auf Seite 207
Dieser Abschnitt enthält Informationen zu den Komponenten der MQI (Message Queue Interface).

„Verbindung zu einem Warteschlangenmanager herstellen und trennen” auf Seite 219

Um WebSphere MQ-Programmierungsservice zu verwenden, 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 228
In diesem Abschnitt finden Sie Informationen zum Öffnen und Schließen von WebSphere MQ-Objekten.

„Nachrichten in eine Warteschlange einreihen” auf Seite 239
In diesem Abschnitt erfahren Sie, wie Nachrichten in eine Warteschlange eingereiht werden.

„Nachrichten aus einer Warteschlange abrufen” auf Seite 255
In diesem Abschnitt erfahren Sie, wie Nachrichten aus einer Warteschlange abgerufen werden.

„Objektattribute abfragen und einstellen” auf Seite 340
Attribute sind Eigenschaften, die die Merkmale eines WebSphere MQ-Objekts beschreiben.

„Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343
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 WebSphere MQ-Anwendungen durch Auslöser starten” auf Seite 350
In diesem Abschnitt erhalten Sie Informationen zu Auslösern und wie Sie IBM WebSphere MQ-Anwendungen mit Auslösern starten.

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.

Bei der Auswahl des Zielwarteschlangenmanagers verwendet der Warteschlangenmanager den Workload-Management-Algorithmus in einem Umlaufverfahren; siehe Lastausgleich.

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 WebSphere 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 und eine Nachricht nicht Teil einer Gruppe ist, wird 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

Die MQOPEN -Optionen MQOO_BROWSE, MQOO_INPUT_*oder MQOO_SET erfordern eine lokale Instanz der Clusterwarteschlange, damit MQOPEN erfolgreich ist.

Für die MQOPEN-Optionen MQOO_OUTPUT, MQOO_BIND_* und MQOO_INQUIRE hingegen ist eine solche lokale Instanz der Clusterwarteschlange nicht erforderlich.

Aufgelöster Name des Warteschlangenmanagers

Wenn ein Warteschlangenmanagername beim Aufruf von MQOPEN aufgelöst wird, wird der aufgelöste Name an die Anwendung zurückgegeben. Verwendet die Anwendung diesen Namen beim nächsten MQOPEN-Aufruf, so kann sich herausstellen, dass die Anwendung über keine Berechtigung für diesen Namen verfügt.

MQPUT, MQPUT1 und Cluster

Bei Angabe von MQOO_BIND_NOT_FIXED im Aufruf MQOPEN entscheidet die Routine zur Auslastungsverwaltung, welches Ziel MQPUT oder MQPUT1 nehmen.

Bei Angabe von MQOO_BIND_NOT_FIXED im Aufruf MQOPEN ruft jeder nachfolgende MQPUT-Aufruf die Routine für Auslastungsverwaltung auf, die dann entscheidet, an welchen Warteschlangenmanager die Nachricht gesendet wird. Die Zieladresse und die Übertragungsroute wird für jede einzelne Nachricht individuell ermittelt. Nachdem eine Nachricht eingereiht wurde, können sich bei Änderungen der Netzbedingungen auch die Zieladresse und die Route ändern. Der Aufruf MQPUT1 wird immer so ausgeführt, als ob die Option MQOO_BIND_NOT_FIXED angegeben wäre, d. h., bei jedem Aufruf wird die Routine zur Auslastungsverwaltung aufgerufen.

Nachdem von der Routine zur Auslastungsverwaltung ein Warteschlangenmanager ausgewählt wurde, führt der lokale Warteschlangenmanager den PUT-Vorgang zu Ende. Die Nachricht kann in verschiedene Warteschlangen eingereiht werden:

1. Wenn das Ziel die lokale Instanz der Warteschlange ist, wird die Nachricht in die lokale Warteschlange eingereiht.

2. Wenn das Ziel ein Warteschlangenmanager in einem Cluster ist, wird die Nachricht in eine Clusterübertragungswarteschlange eingereiht.
3. Wenn das Ziel ein Warteschlangenmanager außerhalb eines Clusters ist, wird die Nachricht in eine Übertragungswarteschlange mit dem Namen des Zielwarteschlangenmanagers eingereiht.

Bei Angabe von MQOO_BIND_ON_OPEN im MQOPEN-Aufruf wird durch nachfolgende MQPUT-Aufrufe die Routine zur Auslastungsverwaltung nicht aufgerufen, da Ziel und Route bereits ausgewählt sind.

MQINQ und Cluster

Welche Clusterwarteschlange abgefragt wird, ist abhängig von den Optionen, die Sie mit MQOO_INQUIRE kombinieren.

Bevor Sie eine Warteschlange abfragen, müssen Sie sie mithilfe des MQOPEN-Aufrufs öffnen und MQOO_INQUIRE angeben.

Zum Abfragen einer Clusterwarteschlange verwenden Sie den MQOPEN-Aufruf und kombinieren andere Optionen mit MQOO_INQUIRE. Welche Attribute abgefragt werden können, hängt davon ab, ob eine lokale Instanz der Clusterwarteschlange vorhanden ist, und davon, wie die Warteschlange geöffnet wird:

- Das Kombinieren von MQOO_BROWSE, MQOO_INPUT_* oder MQOO_SET mit MQOO_INQUIRE erfordert eine lokale Instanz der Clusterwarteschlange, damit das Öffnen erfolgreich ist. In diesem Fall können Sie alle Attribute abfragen, die für lokale Warteschlangen gültig sind.
- Wenn MQOO_OUTPUT mit MQOO_INQUIRE kombiniert und keine der vorhergehenden Optionen angegeben wird, wird eine der folgenden Instanzen geöffnet:
  - Die Instanz auf dem lokalen Warteschlangenmanager, falls vorhanden. In diesem Fall können Sie alle Attribute abfragen, die für lokale Warteschlangen gültig sind.
  - Eine Instanz an einem anderen Ort im Cluster, wenn es keine lokale Warteschlangenmanagerinstanz gibt. In diesem Fall nur können die folgenden Attribute abgefragt werden. Das Attribut QType hat hierbei den Wert MQQT_CLUSTER.
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

Zum Abfragen des DefBind-Attributs einer Clusterwarteschlange verwenden Sie den MQINQ-Aufruf mit dem Selektor MQIA_DEF_BIND. Der zurückgegebene Wert lautet entweder MQBND_BIND_ON_OPEN oder MQBND_BIND_NOT_FIXED oder MQBND_BIND_ON_GROUP. 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 234
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.

Rückgabecodes

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 WebSphere 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.

Auf anderen Plattformen als z/OS wird ein MQCONN- oder MQCONNX-Aufruf ausgegeben, um eine Verbindung mit 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 Rückkehrcode wird nur in den folgenden Fällen zurückgegeben:

- Es ist keine lokale Instanz der Warteschlange vorhanden. Ist eine lokale Instanz vorhanden, verläuft die Ausführung des MQOPEN-Aufrufs erfolgreich, selbst wenn die lokale Instanz für PUT-Vorgänge gesperrt ist.
- Für die Warteschlange ist kein Exit für Clusterauslastung vorhanden, oder der Exit ist zwar vorhanden, er wählt jedoch keine Warteschlangeninstanz aus. (Wenn der Exit für Clusterauslastung eine Warteschlangeninstanz ausgewählt, verläuft der MQOPEN-Aufruf erfolgreich, selbst wenn die Instanz für PUT-Vorgänge gesperrt ist.)

Ist im Aufruf MQOPEN die Option MQOO_BIND_NOT_FIXED angegeben, kann der Aufruf erfolgreich verlaufen, selbst wenn alle Warteschlangen im Cluster für PUT-Vorgänge gesperrt sind. Jedoch kann ein nachfolgender MQPUT-Aufruf fehlschlagen, wenn die Warteschlangen zum Zeitpunkt des Aufrufs noch immer für PUT-Vorgänge gesperrt sind.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Zum Öffnen einer Clusterwarteschlange bzw. zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQOPEN-, ein MQPUT- oder ein MQPUT1-Aufruf ausgegeben. Die Warteschlangendefinition kann nicht korrekt aufgelöst werden, da vom Warteschlangenmanager mit vollständigem Repository eine Antwort benötigt wird, die nicht eintrifft.
2. Ein Aufruf MQOPEN, MQPUT, MQPUT1 oder MQSUB wird für ein Themenobjekt ausgegeben, das PUBSCOPE(ALL) bzw. SUBSCOPE(ALL) angibt. Die Clusterthemendefinition kann nicht ordnungsgemäß aufgelöst werden, weil eine Antwort vom Warteschlangenmanager des vollständigen Repositorys erforderlich ist, aber keine verfügbar ist.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Für eine Clusterwarteschlange wurde ein MQOPEN-, MQPUT- oder MQPUT1-Aufruf ausgegeben. Beim Versuch, eine für das Clustering erforderliche Ressource zu verwenden, tritt ein Fehler auf.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQPUT- oder MQPUT1-Aufruf ausgegeben. Zum Zeitpunkt des Aufrufs befinden sich im Cluster jedoch keine Instanzen dieser Warteschlange mehr. Der MQPUT-Aufruf schlägt fehl, und die Nachricht wird nicht gesendet.

Dieser Fehler kann auftreten, wenn im Aufruf MQOO_BIND_NOT_FIXED zum Öffnen der Warteschlange die Option MQOPEN angegeben wurde, oder die Nachricht mit dem Aufruf MQPUT1 eingereiht wird.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Zum Öffnen einer Clusterwarteschlange bzw. zum Einreihen einer Nachricht in eine Clusterwarteschlange wurde ein MQOPEN-, MQPUT- oder MQPUT1-Aufruf ausgegeben. Der Aufruf wurde aber vom Exit für Clusterauslastung zurückgewiesen.

Clientanwendungen schreiben

Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

Anwendungen können in der WebSphere MQ-Clientumgebung erstellt und ausgeführt werden. Dazu muss die Anwendung erstellt und mit dem verwendeten WebSphere MQ MQI-Client verbunden werden. Die Vorgehensweise hängt dabei von der Plattform und der Programmiersprache ab. Informationen zum Schreiben von Clientanwendungen finden Sie im Abschnitt „Erstellen von Anwendungen für WebSphere MQ-Clients” auf Seite 380.

Sofern bestimmte Bedingungen erfüllt sind, können Sie eine WebSphere MQ-Anwendung sowohl in einer vollständigen WebSphere MQ-Umgebung als auch in einer WebSphere MQ-Clientumgebung einsetzen, ohne dass Änderungen am Code erforderlich sind. Weitere Informationen zur Ausführung Ihrer Anwendungen in der WebSphere MQ-Clientumgebung finden Sie im Abschnitt „Anwendungen in der IBM WebSphere MQ-MQI-Clientumgebung ausführen” auf Seite 382.

Wenn Sie die Message Queue Interface (MQI) zum Schreiben von Anwendungen verwenden, die in einer WebSphere MQ-Clientumgebung ausgeführt werden sollen, müssen Sie während des MQI-Aufrufs einige zusätzliche Steuerelemente implementieren, um sicherzustellen, dass die WebSphere MQ-Anwendungsverarbeitung nicht unterbrochen wird. Weitere Informationen zu diesen Steuerelementen finden Sie im Abschnitt „Message Queue Interface (MQI) in Clientanwendungen verwenden” auf Seite 375.

In den folgenden Abschnitten finden Sie Informationen zum Vorbereiten und Ausführen anderer Anwendungstypen als Clientanwendungen:

- „CICS- und Tuxedo-Anwendungen vorbereiten und ausführen” auf Seite 395
- „MTS-Anwendungen (Microsoft Transaction Server) vorbereiten und ausführen” auf Seite 42
- „WebSphere MQ-JMS-Anwendungen vorbereiten und ausführen” auf Seite 398

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 8

Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„IBM WebSphere MQ-Anwendungen entwerfen” auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„WebSphere MQ-Beispielprogramme” auf Seite 100
In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

„Anwendung zur Warteschlangensteuerung schreiben” auf Seite 206
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.

„Web-Services in WebSphere MQ verwenden” auf Seite 1004
Sie können IBM WebSphere MQ-Anwendungen für Web-Services mithilfe des IBM WebSphere MQ-Transports für SOAP oder der IBM WebSphere MQ-Bridge für HTTP entwickeln.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 295
Beginnen Sie nun, Ihre eigenen WebSphere MQ-Publish/Subscribe-Anwendungen zu entwickeln.

„IBM WebSphere MQ-Anwendung erstellen” auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

„Programmfehler behandeln” auf Seite 581
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.

Message Queue Interface (MQI) in Clientanwendungen verwenden

In diesen Abschnitten wird auf die Unterschiede bei der Erstellung eigener WebSphere MQ-Anwendungen in einer WebSphere MQ MQI-Clientumgebung bzw. in einer vollständigen WebSphere MQ-Warteschlangenmanager-Umgebung eingegangen.

Überlegen Sie sich bei der Konzipierung eigener Anwendungen, welche Kontrollmechanismen bei einem MQI-Aufruf erforderlich sind, um sicherzustellen, dass die WebSphere MQ-Anwendungsverarbeitung ungestört verläuft.

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.

Auf allen Plattformen mit Ausnahme von z/OS können Sie die maximale Nachrichtenlänge eines Warteschlangenmanagers heraufsetzen. Genaue Informationen hierzu finden Sie in der Beschreibung von 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 WebSphere 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.

Client- oder servercodierte Zeichensatzkennung (CCSID) auswählen

Verwenden Sie die lokale CCSID des Clients. 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 an den Client-Stub übertragenen Daten müssen in der lokalen CCSID, codiert für den WebSphere MQ-Client, vorliegen. Falls für den verbundenen Warteschlangenmanager eine Konvertierung der Daten erforderlich ist, wird die Konvertierung vom Clientunterstützungscode des Warteschlangenmanagers ausgeführt.

Der Java-Client der Version V7 kann die Konvertierung auch selbst durchführen, wenn der Warteschlangenmanager hierzu nicht in der Lage ist. Siehe „ WebSphere MQ Classes for Java-Clientverbindungen” auf Seite 709

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:

- Unter Windows:

```
SET MQCCSID=850
```

- Unter UNIX-Systemen:

```
export MQCCSID=850
```

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 die Anwendung mehrere PUT-Operationen ausführt, die anschließend an den Nachrichtendeskriptor (MQMD) WebSphere MQ-Header enthalten, beachten Sie, dass die CCSID und die Codierungsfelder der MQMD nach Abschluss der ersten PUT-Operation überschrieben werden.

Nach der ersten PUT-Operation enthalten diese Felder den vom verbundenen Warteschlangenmanager zur Konvertierung der WebSphere 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 WebSphere MQ besteht eine der Aufgaben des Warteschlangenmanagers auch in der Synchronisationspunktsteuerung innerhalb der Anwendung. Eine auf dem WebSphere MQ-Basisclient ausgeführte Anwendung kann die Aufrufe MQCMIT und MQBACK ausgeben, die Synchronisationspunktsteuerung beschränkt sich jedoch auf die MQI-Ressourcen. Der WebSphere 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 WebSphere MQ-Produkten bereitgestellten Transaktionsmonitor oder auch Transaktionsmonitor wie CICS verwenden. Bei Basisclientanwendungen können Sie keinen Transaktionsmonitor verwenden.

Bei einem erweiterten transaktionsorientierten WebSphere MQ-Client können Sie einen externen Transaktionsmanager verwenden. Siehe auch Was ist ein erweiterter transaktionsorientierter Client? .

Vorauslesen in einer Clientanwendung verwenden

Sie können Vorauslesen auf einem Client verwenden, damit nicht persistente Nachrichten an einen Client gesendet werden können, ohne dass die Clientanwendung die Nachrichten anfordern muss.

Wenn ein Client eine Nachricht von einem Server benötigt, sendet er eine Anforderung an den Server. Für jede von ihm verarbeitete Nachricht sendet der Client eine eigene Anforderung. Zur Verbesserung der Leistung eines Clients, der nicht persistente Nachrichten verarbeitet, kann er für die Verwendung von Vorauslesen konfiguriert und es so vermieden werden, dass er diese Anforderungsnachrichten senden muss. Mit Vorauslesen können Nachrichten an einen Client gesendet werden, ohne dass eine Anwendung sie anfordern muss.

Durch die Verwendung von Vorauslesen kann die Leistung verbessert werden, wenn nicht persistente Nachrichten von einer Clientanwendung verarbeitet werden. Diese Leistungsverbesserung ist für MQI- und JMS-Anwendungen verfügbar. Clientanwendungen, die MQGET oder asynchrone Verarbeitung verwenden, profitieren von den Leistungsverbesserungen, wenn sie nicht persistente Nachrichten verarbeiten.

Wenn Sie MQOPEN mit MQOO_READ_AHEAD aufrufen, aktiviert der WebSphere MQ-Client Vorauslesen nur unter bestimmten Bedingungen. Zu diesen Bedingungen gehören:

- Client und ferner Warteschlangenmanager müssen beide die WebSphere MQ Version 7 oder höher haben.
- Die Clientanwendung muss kompiliert und mit den WebSphere MQ-Clientbibliotheken mit Thread 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 274.

Die Konfiguration für das Vorauslesen wird durch drei Attribute, MaximumSize, PurgeTime und UpdatePercentage, gesteuert, die in der MessageBuffer-Zeilengruppe der WebSphere MQ-Clientkonfigurationsdatei angegeben werden.

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 121.

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/SSL-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 vom Kanal *nicht* 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 422.

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 WebSphere MQ-Client auch eine Verbindung zu einem Warteschlangenmanager herstellen, der keine gemeinsam genutzten Verbindungskennungen unterstützt.

Weitere Informationen finden Sie unter „MQCONNX verwenden” auf Seite 380.

Erstellen von Anwendungen für WebSphere MQ-Clients

Anwendungen können in der WebSphere MQ-Clientumgebung erstellt und ausgeführt werden. Dazu muss die Anwendung erstellt und mit dem verwendeten WebSphere MQ MQI-Client verbunden 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 47. In Clientumgebungen unterstützte Programmiersprachen

Clientplattform	C	C + +	COBOL	pTAL	RPG	Visual Basic
AIX	Ja	Ja	Ja			

Tabelle 47. In Clientumgebungen unterstützte Programmiersprachen (Forts.)						
Clientplattform	**C**	**C + +**	**COBOL**	**pTAL**	**RPG**	**Visual Basic**
HP Integrity NonStop Server	Ja		Ja	Ja		
HP-UX	Ja	Ja	Ja			
Linux	Ja	Ja	Ja			
Solaris	Ja	Ja	Ja			
Windows	Ja	Ja	Ja			Ja

Hinweise zum Verbinden oder Erstellen von Clientanwendungen, die mit diesen Programmiersprachen geschrieben wurden, finden Sie in den entsprechenden Abschnitten.

C-Anwendungen mit dem WebSphere MQ-MQI-Client-Code verknüpfen

Nachdem Sie die WebSphere MQ-Anwendung, die Sie auf dem WebSphere MQ-Client ausführen möchten, geschrieben haben, müssen Sie diese noch mit einem Warteschlangenmanager verknüpfen.

Sie können die Anwendung auf zwei Arten mit einem Warteschlangenmanager verknüpfen:

1. Direkt, wobei sich der Warteschlangenmanager auf derselben Workstation befinden muss wie die Anwendung.
2. Über eine Clientbibliotheksdatei, die Ihnen Zugriff auf Warteschlangenmanager auf derselben oder auf einer anderen Workstation gibt.

WebSphere MQ stellt für jede Umgebung eine Clientbibliotheksdatei bereit:

AIX
Bibliothek libmqic.a für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.a für Thread-Anwendungen

HP-UX
Bibliothek libmqic.sl für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.sl für Thread-Anwendungen

Linux
Bibliothek libmqic.so für Nicht-Thread-Anwendungen bzw. Bibliothek libmqic_r.so für Thread-Anwendungen

Solaris
libmqic.so.

Wenn Sie die Programme auf einer Workstation verwenden möchten, auf der nur der WebSphere MQ-Client installiert ist, müssen Sie die Programme neu kompilieren, um sie mit der Clientbibliothek zu verknüpfen:

```
$ /opt/SUNWspro/bin/cc -o <prog> <prog> c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Die Parameter müssen in der angegebenen Reihenfolge eingegeben werden.

Windows
MQIC32.LIB.

C++-Anwendungen mit dem WebSphere MQ-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 WebSphere MQ C++-Programme erstellen.

Ausführliche Informationen zur Verwendung der Programmiersprache C++ finden Sie im Abschnitt C++ verwenden.

Verknüpfen von COBOL-Anwendungen mit dem IBM WebSphere MQ-Client-Code

Nachdem Sie die COBOL-Anwendung, die Sie auf dem IBM WebSphere MQ-Client ausführen möchten, geschrieben haben, müssen Sie sie noch mit der richtigen Bibliothek verknüpfen.

IBM WebSphere MQ bietet für jede Umgebung eine Clientbibliotheksdatei:

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
Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.sl und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.sl.

Linux
Verknüpfen Sie eine Nicht-Thread-COBOL-Anwendung mit der Bibliothek libmqicb.so und eine Thread-COBOL-Anwendung mit der Bibliothek libmqicb_r.so.

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
Verknüpfen Sie Ihren Anwendungscode mit der MQICCBB-Bibliothek für die 32-Bit-Version von COBOL. Der IBM WebSphere MQ-Client für Windows unterstützt kein 16-Bit-COBOL.

Verknüpfen von Visual Basic-Anwendungen mit dem WebSphere MQ-Client-Code

Sie können Visual Basic-Anwendungen mit dem WebSphere MQ MQI-Client-Code unter Windowsverknüpfen.

Ihre Visual Basic-Anwendung können Sie mit folgenden Include-Dateien verknüpfen:

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
PCF-Befehle

CMQXB.bas
Kanäle

Geben Sie im Visual Basic-Compiler für den Client `mqtype=2` an, um sicherzustellen, dass die Client-DLL durch die automatische Auswahl korrekt ausgewählt wird:

MQIC32.dll
Windows 2000, Windows XP und Windows 2003

Anwendungen in der IBM WebSphere MQ-MQI-Clientumgebung ausführen

Sie können eine IBM WebSphere MQ-Anwendung in einer vollständigen IBM WebSphere MQ-Umgebung und in einer IBM WebSphere MQ MQI-Clientumgebung ohne Änderungen am Code ausführen, sofern bestimmte Bedingungen erfüllt sind.

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.
- Die Anwendung muss keine der Ausnahmen verwenden, die unter Welche Anwendungen werden auf einem IBM WebSphere MQ MQI-Client ausgeführt? aufgelistet sind.

Anmerkung: Die Bibliotheken, die bei der Programmverbindung verwendet werden, geben die Umgebung vor, in der die Anwendung ausgeführt werden muss.

Bei Verwendung der IBM WebSphere MQ-MQI-Clientumgebungen sollten Sie Folgendes bedenken:

- Jede in der IBM WebSphere MQ-MQI-Clientumgebung aktive Anwendung verfügt über eigene Verbindungen zum Server. 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 WebSphere MQ MQI-Clientanwendungen mit Warteschlangenmanagern verbinden

Es gibt verschiedene Möglichkeiten für eine in einer IBM WebSphere MQ-MQI-Clientumgebung aktive Anwendung, eine Verbindung zu einem Warteschlangenmanager herzustellen: über Umgebungsvariablen, die MQCNO-Struktur oder eine Clientdefinitionstabelle.

Wenn eine in einer IBM WebSphere 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 WebSphere MQ-Client einen MQCONNX-Aufruf aus, sucht die MQI-Clientbibliothek in der folgenden Reihenfolge nach den Clientkanalinformationen:

1. Anhand des Inhalts des Felds *ClientConnOffset* oder *ClientConnPtr* der MQCNO-Struktur (sofern vorhanden). Diese Felder geben die Kanaldefinitionsstruktur (MQCD) an, die als Definition für den Clientverbindungskanal verwendet werden soll. Verbindungsdetails können mithilfe eines Pre-Connect-Exits überschrieben werden. Weitere Informationen finden Sie unter „Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren” auf Seite 450.
2. Wurde die Umgebungsvariable MQSERVER gesetzt, wird der von ihr angegebene Kanal verwendet.
3. Wenn eine Datei mqclient.ini definiert ist und ein Attribut 'ServerConnectionParms' enthält, wird der von ihr definierte Kanal verwendet. Weitere Informationen finden Sie in den Abschnitten Client mithilfe einer Konfigurationsdatei konfigurieren und CHANNELS-Zeilengruppe der Clientkonfigurationsdatei.
4. Wurden die Umgebungsvariablen MQCHLLIB und MQCHLTAB gesetzt, wird die Clientkanal-Definitionstabelle verwendet, auf die diese Variablen verweisen.
5. Wenn eine Datei mqclient.ini 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 Clientdefinitionstabelle ergebnislos, verwendet der Client die folgenden Pfade:
   - UNIX and Linux-Systeme: /var/mqm/AMQCLCHL.TAB
   - Windows: C:\Program Files\IBM\Websphere MQ\amqclchl.tab

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.

Wenn Ihre Anwendung den Rückgabecode MQRC_Q_MGR_NOT_AVAILABLE mit der Fehlernachricht `AMQ9517 - File damaged` (Datei beschädigt) im Fehlerprotokoll zurückgibt, finden Sie weitere Informationen im Abschnitt Migration und Clientkanal-Definitionstabellen (CCDT).

Zugehörige Konzepte

Definitionstabelle für den Clientkanal

Zugehörige Tasks

Verbindungen zwischen dem Server und dem Client konfigurieren

Zugehörige Verweise

MQSERVER

MQCHLLIB

MQCHLTAB

Clientanwendungen mithilfe von Umgebungsvariablen mit Warteschlangenmanagern verbinden

Clientkanalinformationen können einer in einer Clientumgebung ausgeführten Anwendung durch die Umgebungsvariablen MQSERVER, MQCHLLIB und MQCHLTAB bereitgestellt werden.

Einzelheiten zu diesen Variablen finden Sie in den Abschnitten MQSERVER, MQCHLLIB und MQCHLTAB.

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 in einem MQCONNX-Aufruf 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.

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.

Anmerkung: Die gleiche Datei kann auch von mehreren IBM WebSphere MQ-Clients verwendet werden. Mit den IBM WebSphere MQ-Umgebungsvariablen MQCHLLIB und MQCHLTAB legen Sie fest, auf welche Version dieser Datei Sie zugreifen. Informationen zu Umgebungsvariablen finden Sie im Abschnitt WebSphere MQ-Umgebungsvariablen verwenden.

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. Für einen erweiterten transaktionsorientierten Client oder einen Java-Client in WebSphere Application Server kann es zum Beispiel notwendig sein, eine Verbindung zu einer vorhersehbaren Instanz eines Warteschlangenmanagers herzustellen. Die automatische Wiederherstellung einer Clientverbindung wird von WebSphere MQ-Klassen für 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 70 auf Seite 386 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.

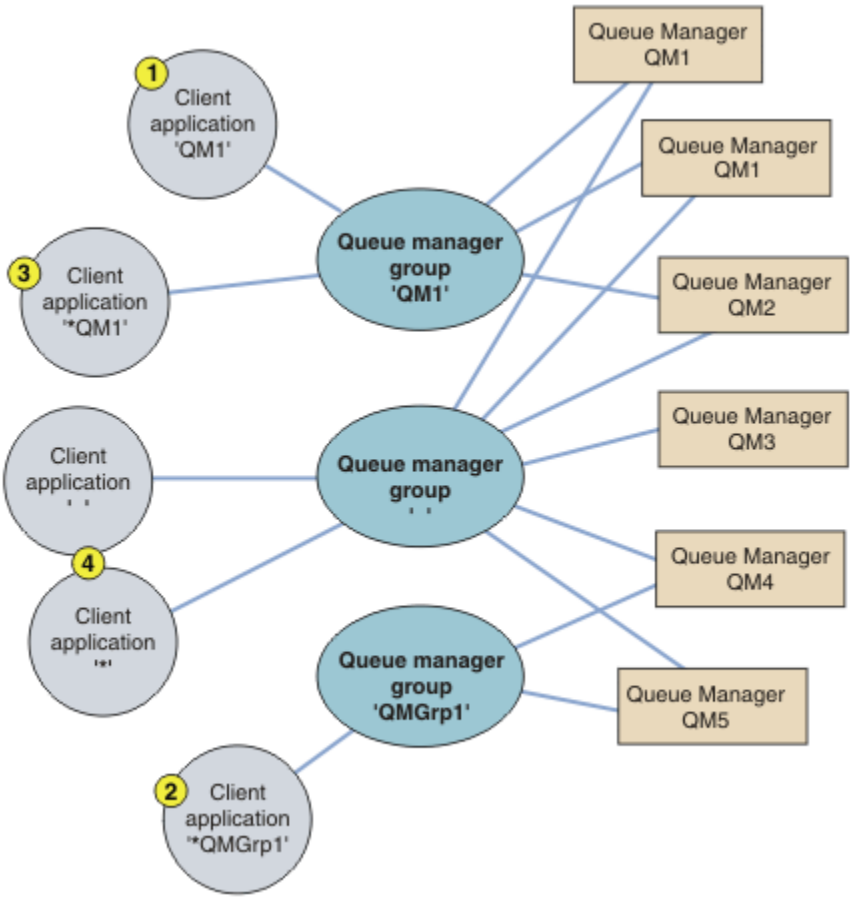


Abbildung 70. WS-Manager-Gruppen

Vier Beispiele für die Verwendung dieser Clientverbindungstabelle werden mithilfe der nummerierten Clientanwendungen in Abbildung 70 auf Seite 386 beschrieben.

1. Im ersten Beispiel übergibt die Clientanwendung einen Warteschlangenmanagernamen, `QM1`, als **`QmgrName`**-Parameter an ihren MQI-Aufruf `MQCONN` oder `MQCONNX`. Der WebSphere MQ-Client-Code wählt die entsprechende Warteschlangenmanager-Gruppe QM1 aus. Die Gruppe enthält drei Verbindungskanäle und der WebSphere MQ-Client versucht, eine Verbindung zu `QM1` herzustellen. Dazu verwendet er nacheinander jeden dieser Kanäle, bis er ein WebSphere MQ-Listener für die Verbindung an einem aktiven Warteschlangenmanager namens `QM1` findet.

   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 WebSphere MQ-Client wählt die entsprechende Warteschlangenmanager-Gruppe QMGrp1 aus. Diese Gruppe enthält zwei Clientverbindungskanäle und der WebSphere MQ-Client versucht, eine Verbindung zu einem *beliebigen* Warteschlangenmanager herzustellen. Dazu verwendet er nacheinander jeden dieser Kanäle. In diesem Beispiel muss der WebSphere MQ-Client erfolgreich eine Verbindung herstellen, wobei der Name des Warteschlangenmanagers für die Verbindung unerheblich ist.

   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 und versuchen Sie dann, 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 Warteschlangenmanager-Gruppe, aber legen Sie für jede Clientkanaldefinition das Attribut **AFFINITY**(PREFERRED) fest.

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 WebSphere MQ-Client wiederholt Verbindungen wahlfrei über Warteschlangenmanager und verteilt die Verbindungswiederholungen zeitlich.

Verschieben von Warteschlangenmanagern ohne Ändern von Clientcode.

Die Definitionstabelle für Clientkanäle isoliert die Clientanwendung von der Position des Warteschlangenmanagers.

Sie haben die Möglichkeit, die Clientverbindungstabelle auf jeden Client zu verteilen oder die Definitionstabelle für Clientkanäle in einem gemeinsam genutzten Dateisystem abzulegen, damit jeder Client darin nachschlagen kann. Sie können aber auch die programmgestützte Version der Definitionstabelle für Clientkanäle verwenden, die im MQI-Aufruf MQCONNX unterstützt wird, und einen Service aufrufen, um die Tabelle an die Clientanwendung zu übergeben.

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.

Verbindung zu Gruppe mit gemeinsamer Warteschlange herstellen

Sie können eine Anwendung mit einem Warteschlangenmanager verbinden, der Teil einer Gruppe mit gemeinsamer Warteschlange ist. Dazu 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.

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. MQCONN-Aufrufe, deren Ergebnis eine Verbindung zu einem der verschiedenen Warteschlangenmanager ist, müssen dem Namen des Warteschlangenmanagers als Präfix einen Stern voranstellen, wie 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 WebSphere MQ-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 WebSphere MQ-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 selben WebSphere MQ-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.

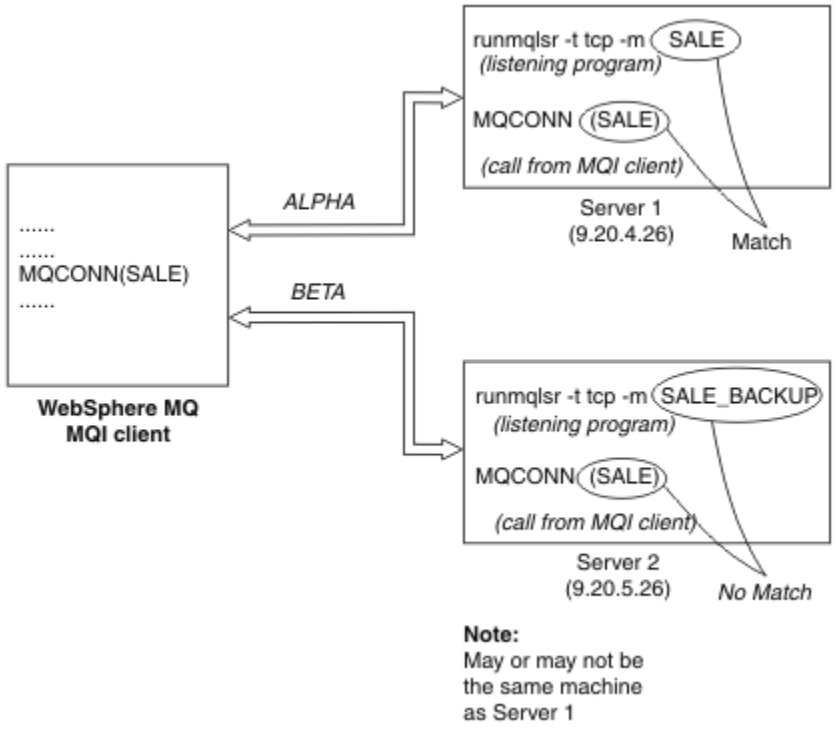


Abbildung 71. Beispiel für MQCONN-Aufruf

Die Kanäle in diesen Beispielen sind wie folgt definiert:

Definitionen für SALE:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('WebSphere MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('WebSphere MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definition für SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')
```

Hier eine Zusammenfassung der Clientkanaldefinitionen:

Name	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

Was die MQCONN-Beispiele veranschaulichen
Die Beispiele veranschaulichen die Verwendung mehrerer Warteschlangenmanager als Ausweichsystem.

Nehmen wir an, die Kommunikation mit Server 1 ist vorübergehend unterbrochen. In einem solchen Fall ist die Verwendung mehrerer Warteschlangenmanager als Ausweichsystem sehr nützlich. Dieses soll hier veranschaulicht werden.

Jedes Beispiel behandelt einen anderen MQCONN-Aufruf und erläutert die Vorgänge im jeweiligen Fall, wobei die folgenden Regeln gelten:

1. Die Clientkanal-Definitionstabelle (CCDT) wird in alphabetischer Reihenfolge der Kanalnamen nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen (QMNAME) durchsucht.
2. Bei einer Übereinstimmung wird die Kanaldefinition verwendet.
3. Es wird versucht, den Kanal zu dem durch den Verbindungsnamen (CONNAME) angegebenen System zu starten. Bei Erfolg wird die Anwendung fortgesetzt. Für eine erfolgreiche Verbindung ist Folgendes erforderlich:
   - Ein aktives Empfangsprogramm auf dem Server.
   - Das Empfangsprogramm muss mit dem Warteschlangenmanager verbunden sein, zu dem der Client eine Verbindung herstellen will (sofern angegeben).
4. Falls der Versuch, den Kanal zu starten, fehlschlägt und die Clientkanal-Definitionstabelle noch weitere Einträge enthält (im Beispiel enthält sie zwei Einträge), wird die Tabelle nach einer weiteren Übereinstimmung durchsucht. Wird eine solche gefunden, so beginnt die Verarbeitung wieder bei Schritt 1.
5. Wird keine Übereinstimmung gefunden bzw. enthält die Clientkanal-Definitionstabelle keine weiteren Einträge mehr und der Kanal konnte nicht gestartet werden, so kann die Anwendung keine Verbindung herstellen. In diesem Fall gibt der MQCONN-Aufruf einen entsprechenden Beendigungs- und Ursachencode zurück. Auf Grundlage der zurückgegebenen Codes kann die Anwendung geeignete Schritte unternehmen.

Beispiel 1: Name des Warteschlangenmanagers enthält Stern (*)
In diesem Beispiel spielt es für die Anwendung keine Rolle, mit welchem Warteschlangenmanager sie verbunden wird. Die Anwendung gibt einen MQCONN-Aufruf mit einem Warteschlangenmanagernamen aus, der einen Stern (*) enthält. Daraufhin wird ein geeigneter Kanal ausgewählt.

Die Anwendung gibt folgenden Aufruf aus:

```
MQCONN (*SALE)
```

Den Regeln entsprechend geschieht nun Folgendes:

1. Die Clientkanal-Definitionstabelle (CCDT) wird nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen SALE durchsucht.
2. Die Kanaldefinitionen für ALPHA und BETA werden gefunden.
3. Wenn ein Kanal den CLNTWGHT-Wert 0 hat, wird dieser Kanal ausgewählt. Wenn beide Kanäle den CLNTWGHT-Wert 0 haben, wird Kanal ALPHA ausgewählt, da er in alphabetischer Reihenfolge der erste Kanal ist. Wenn beide Kanäle einen CLNTWGHT-Wert ungleich null haben, wird der Kanal nach seiner Gewichtung ausgewählt.
4. Es wird versucht, den Kanal zu starten.
5. Wenn Kanal BETA ausgewählt wurde, ist der Start erfolgreich.
6. Wenn Kanal ALPHA ausgewählt wurde, ist der Start nicht erfolgreich, da die Kommunikationsverbindung unterbrochen ist. In diesem Fall werden folgende Schritte ausgeführt:
   a. Der einzige andere Kanal für den Warteschlangenmanager SALE ist BETA.
   b. Es wird versucht, diesen Kanal zu starten - mit Erfolg.
7. Die Überprüfung, ob ein Empfangsprogramm aktiv ist, ist erfolgreich. Dieses ist zwar nicht mit dem Warteschlangenmanager SALE verbunden, da der MQI-Aufrufparameter jedoch einen Stern (*) im Warteschlangenmanagernamen enthält, wird diesbezüglich keine Überprüfung vorgenommen. Die Anwendung wird mit dem Warteschlangenmanager SALE_BACKUP verbunden und die Verarbeitung kann fortgesetzt werden.

Beispiel 2: Name des WS-Managers angegeben
In diesem Beispiel muss die Anwendung eine Verbindung zu einem bestimmten Warteschlangenmanager herstellen. Die Anwendung gibt einen MQCONN-Aufruf mit dem Namen dieses Warteschlangenmanagers aus. Daraufhin wird ein geeigneter Kanal ausgewählt.

Die Anwendung fordert eine Verbindung zu einem bestimmten Warteschlangenmanager namens SALE an und gibt dazu folgenden MQI-Aufruf aus:

```
MQCONN (SALE)
```

Den Regeln entsprechend geschieht nun Folgendes:

1. Die Clientkanal-Definitionstabelle (CCDT) wird in alphabetischer Reihenfolge der Kanalnamen nach dem im MQCONN-Aufruf angegebenen Warteschlangenmanagernamen SALE durchsucht.
2. Die erste Kanaldefinition, die übereinstimmt, ist ALPHA.
3. Es wird versucht, diesen Kanal zu starten - *erfolglos*, da die Kommunikationsverbindung unterbrochen ist.
4. Die Clientkanal-Definitionstabelle (CCDT) wird erneut nach dem Warteschlangenmanagernamen SALE durchsucht, wobei der Kanal BETA gefunden wird.
5. Es wird versucht, diesen Kanal zu starten - mit Erfolg.
6. Die Überprüfung, ob ein Empfangsprogramm aktiv ist, ist erfolgreich, jedoch ist dieses Empfangsprogramm nicht mit dem Warteschlangenmanager SALE verbunden.
7. Die Definitionstabelle für Clientkanäle enthält keine weiteren Einträge mehr. Die Anwendung kann die Verarbeitung nicht fortsetzen und erhält den Rückgabecode MQRC_Q_MGR_NOT_AVAILABLE.

Beispiel 3. Name des Warteschlangenmanagers ist leer oder ein Stern (*)
In diesem Beispiel spielt es für die Anwendung keine Rolle, mit welchem Warteschlangenmanager sie verbunden wird. Die Anwendung gibt einen MQCONN-Aufruf ohne Warteschlangenmanagername bzw. mit einem Stern als Warteschlangenmanagername aus. Daraufhin wird ein geeigneter Kanal ausgewählt.

Dieser Aufruf wird genauso behandelt wie der Aufruf in „Beispiel 1: Name des Warteschlangenmanagers enthält Stern (*)" auf Seite 392.

Anmerkung: Wird die Anwendung allerdings in einer anderen Umgebung als der WebSphere MQ-Clientumgebung 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 das Empfangsprogramm 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, der einen leeren Warteschlangenmanagernamen anfordert.
3. Der nächste Eintrag gehört zum Kanalnamen BETA.
4. Die `queue manager name` in der Definition ist SALE. Wiederum entspricht dies *nicht* dem MQCONN-Aufrufparameter, der einen leeren Warteschlangenmanagernamen anfordert.

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 WebSphere MQ-Anwendungen auf WebSphere MQ-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 andere Betriebssystem des Servers seine Standarddefinitionen und der Prozess schlägt fehl.

Auslösemonitor

Der von Nicht-z/OS WebSphere MQ -Produkten bereitgestellte Auslösemonitor wird in den Clientumgebungen für UNIX-, Linux -und Windows -Systeme ausgeführt.

Führen Sie zur Ausführung des Auslösemonitors einen der folgenden Befehle aus:

- Windows Linux UNIX Auf Windows-, UNIX-und Linux -Plattformen:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

Die Standardinitialisierungswarteschlange ist die Warteschlange SYSTEM.DEFAULT.INITIATION.QUEUE auf dem Standardwarteschlangenmanager. Die Initialisierungswarteschlange ist die Warteschlange, die der Auslösemonitor nach Auslösenachrichten durchsucht. Danach ruft er die entsprechenden Programme für die Auslösenachrichten auf. Dieser Auslösemonitor unterstützt den Standardanwendungstyp und ist identisch mit `runmqtrm`, mit der Ausnahme, dass er die Clientbibliotheken verknüpft.

Die vom Auslösemonitor erstellte Befehlszeichenfolge besteht aus folgenden Elementen:

1. Die Anwendungs-ID (*ApplicId*) aus der relevanten Prozessdefinition. Bei *ApplicId* handelt es sich um den Namen des auszuführenden Programms in dem Format, in dem er in der Befehlszeile eingegeben wird.
2. Die MQTMC2-Struktur aus der Initialisierungswarteschlange, eingeschlossen in Anführungszeichen. Eine Befehlsfolge mit exakt der dargestellten Zeichenfolge wird in doppelten Anführungszeichen gestartet, damit der Systembefehl diese als einen Parameter akzeptiert.
3. Die Anwendungs-ID (*EnvrData*) aus der relevanten Prozessdefinition.

Erst nach Abschluss der gestarteten Anwendung durchsucht der Auslösemonitor die Initialisierungswarteschlange wieder nach einer weiteren Nachricht. Falls die Anwendungsverarbeitung lange dauert, kommt der Auslösemonitor daher unter Umständen den in der Warteschlange auflaufenden Auslösenachrichten nicht mehr nach. In einer solchen Situation haben Sie zwei Möglichkeiten:

1. Richten Sie weitere Auslösemonitore ein.

   Bei mehreren Auslösemonitoren können Sie die maximale Anzahl der gleichzeitig ausführbaren Anwendungen eingrenzen.

2. Führen Sie die gestarteten Anwendungen im Hintergrund aus.

   Bei einer Hintergrundausführung grenzt WebSphere MQ die Anzahl der ausführbaren Anwendungen nicht ein.

Zum Ausführen der gestarteten Anwendung im Hintergrund auf UNIX and Linux -Systemen müssen Sie am Ende der *EnvrData* der Prozessdefinition ein Et-Zeichen (&) einfügen.

CICS-Anwendungen (Nicht-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 sie in der Datei CICSENV.CMD angegeben werden.

WebSphere MQ-Anwendungen können ohne Codeänderung in einer WebSphere MQ-Serverumgebung oder auf einem WebSphere MQ-Client ausgeführt werden. In einer WebSphere MQ-Serverumgebung kann CICS als Synchronisationspunktkoordinator fungieren und Sie verwenden EXEC CICS SYNCPOINT und EXEC CICS SYNCPOINT ROLLBACK anstelle von MQCMIT und MQBACK. Wenn eine CICS-Anwendung lediglich als Client neu verbunden wird, dann geht die Synchronisationspunktunterstützung verloren. MQCMIT und MQBACK müssen für die Anwendung benutzt werden, die auf einem WebSphere MQ-MQI-Client ausgeführt wird.

CICS- und Tuxedo-Anwendungen vorbereiten und ausführen

Für die Ausführung von CICS- und Tuxedo-Anwendungen als Clientanwendungen werden andere Bibliotheken als für ihre Ausführung als Serveranwendungen verwendet. Ebenso unterscheidet sich auch die Benutzer-ID, unter denen die Anwendungen aktiv sind.

Zur Vorbereitung von CICS- und Tuxedo-Anwendungen für die Ausführung als WebSphere MQ-MQI-Clientanwendungen befolgen Sie die Anweisungen in Erweiterten transaktionsorientierten Client konfigurieren.

Allerdings wird bei den Informationen zur Vorbereitung von CICS- und Tuxedo-Anwendungen (einschließlich der Musterprogramme, die zusammen mit WebSphere MQ geliefert werden) davon ausgegangen, dass die Anwendungen für die Ausführung auf einem WebSphere MQ-Serversystem vorbereitet werden. Daher beziehen sich die Informationen ausschließlich auf WebSphere MQ-Bibliotheken, die auf einem Serversystem verwendet werden sollen. Wenn Sie dagegen Clientanwendungen vorbereiten, müssen Sie Folgendes beachten:

- Verwenden Sie für das von Ihrer Anwendung verwendete programmiersprachenbezogene Binden die entsprechende Clientsystembibliothek. Für Anwendungen, die mit C auf AIX-, HP-UX- oder Solaris-Systemen erstellt wurden, müssen Sie beispielsweise statt 'libmqm' die Bibliothek 'libmqic' verwenden. Auf Windows-Systemen muss statt 'mqm.lib' die Bibliothek 'mqic.lib' verwendet werden.
- Anstatt der in Tabelle 48 auf Seite 396 (für AIX, HP-UX und Solaris) und Tabelle 49 auf Seite 396 (für Windows-Systeme) aufgeführten Serversystembibliotheken müssen Sie die entsprechenden Clientsystembibliotheken verwenden. Ist eine Serversystembibliothek nicht in diesen Tabellen aufgeführt, verwenden Sie die entsprechende Bibliothek für Clientsysteme.

Tabelle 48. Clientsystembibliotheken unter AIX, HP-UX und Solaris	
Bibliothek für ein WebSphere MQ-Serversystem	**Entsprechende Bibliothek für ein WebSphere MQ-Clientsystem**
libmqmxa	libmqcxa

Tabelle 49. Clientsystembibliotheken auf Windows-Systemen	
Bibliothek für ein WebSphere MQ-Serversystem	**Entsprechende Bibliothek für ein WebSphere MQ-Clientsystem**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

Von einer Clientanwendung verwendete Benutzer-ID

Bei der Ausführung einer WebSphere MQ-Serveranwendung unter CICS wird in der Regel vom CICS-Benutzer auf die Benutzer-ID der Transaktion umgeschaltet. Bei der Ausführung einer WebSphere MQ-MQI-Clientanwendung unter CICS hingegen wird die privilegierte CICS-Berechtigung beibehalten.

CICS- und Tuxedo-Beispielprogramme

CICS- und Tuxedo-Beispielprogramme zur Verwendung auf AIX-, HP-UX-, Solaris- und Windows-Systemen.

In Tabelle 50 auf Seite 396 finden Sie eine Liste der CICS- und Tuxedo-Beispielprogramme, die zur Verwendung auf AIX-, HP-UX- und Solaris-Clientsystemen bereitgestellt werden. In Tabelle 51 auf Seite 396 werden die gleichen Informationen für Windows-Clientsysteme aufgeführt. 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 „CICS-Transaktionsbeispiel” auf Seite 124 und „TUXEDO; Beispielprogramme” auf Seite 165.

Tabelle 50. Beispielprogramme für AIX-, HP-UX- und Solaris-Clientsysteme		
Beschreibung	**Quelle**	**Ausführbares Modul**
CICS-Programm	amqscic0.ccs	amqscicc
Headerdatei für das CICS-Programm	amqscih0.h	-
Tuxedo-Clientprogramm für das Einreihen von Nachrichten	amqstxpx.c	-
Tuxedo-Clientprogramm für das Abrufen von Nachrichten	amqstxgx.c	-
Tuxedo-Serverprogramm für die beiden Clientprogramme	amqstxsx.c	-
UBBCONFIG-Datei für die Tuxedo-Programme	ubbstxcx.cfg	-
Datei mit den Feldtabellen für die Tuxedo-Programme	amqstxvx.flds	-
Datei mit den Ansichtsdefinitionen für die Tuxedo-Programme	amqstxvx.v	-

Tabelle 51. Beispielprogramme für Windows-Clientsysteme		
Beschreibung	**Quelle**	**Ausführbares Modul**
CICS-Transaktion	amqscic0.ccs	amqscicc

Tabelle 51. Beispielprogramme für Windows-Clientsysteme (Forts.)		
Beschreibung	**Quelle**	**Ausführbares Modul**
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	-

Fehlernachricht AMQ5203, modifiziert für CICS- und Tuxedo-Anwendungen

Wenn Sie CICS- oder Tuxedo-Anwendungen mit einem erweiterten transaktionsorientierten Client verwenden, werden möglicherweise Standarddiagnosenachrichten angezeigt. Eine dieser Nachrichten wurde für die Verwendung mit einem erweiterten transaktionsorientierten Client modifiziert.

Die Nachrichten, die in die Fehlerprotokolldateien von WebSphere 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.

MTS-Anwendungen (Microsoft Transaction Server) vorbereiten und ausführen

Um eine MTS-Anwendung für ihre Ausführung als WebSphere MQ-Clientanwendung vorzubereiten, müssen Sie entsprechend den Anweisungen für Ihre Umgebung vorgehen.

Allgemeine Informationen zur Entwicklung von MTS-Awendungen (Microsoft Transaction Server), die auf WebSphere MQ-Ressourcen zugreifen, finden Sie im WebSphere MQ Help Center im Abschnitt über MTS.

Um eine MTS-Anwendung für ihre Ausführung als WebSphere MQ-Clientanwendung vorzubereiten, führen Sie für jede Komponente der Anwendung einen der folgenden Schritte aus:

- Verwendet die Komponente Bindungen der Programmiersprache C für die MQI, müssen Sie entsprechend den Anweisungen im Abschnitt „C-Programme unter Windows vorbereiten” auf Seite 487 vorgehen, die Komponente jedoch mit der Bibliothek 'mqicxa.lib' verbinden, nicht mit 'mqic.lib'.
- Verwendet die Komponente die C++-Klassen von WebSphere MQ, müssen Sie entsprechend den Anweisungen im Thema „C++-Programme unter Windows erstellen” auf Seite 693 vorgehen, die Komponente jedoch mit der Bibliothek 'imqx23vn.lib' verbinden, 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 unter Windows vorbereiten” auf Seite 491 vorgehen, bei der Definition des Visual Basic-Projekts im Feld **Argumente für bedingte Kompilierung** jedoch `MqType=3` eingeben.
- Verwendet die Komponente MQAX (WebSphere MQ Automation Classes for ActiveX), müssen Sie die Umgebungsvariable GMQ_MQ_LIB mit dem Wert `mqic32xa.dll` definieren.

  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.

WebSphere MQ-JMS-Anwendungen vorbereiten und ausführen

Sie können WebSphere MQ-JMS-Anwendungen im Clientmodus ausführen, wobei WebSphere Application Server als Transaktionsmanager fungiert. Möglicherweise werden bestimmte Warnhinweise angezeigt.

Um WebSphere MQ-JMS-Anwendungen mit WebSphere Application Server als Transaktionsmanager im Clientmodus vorzubereiten und auszuführen, gehen Sie entsprechend den Anweisungen im Abschnitt „WebSphere MQ-Klassen für JMS verwenden” auf Seite 759 vor.

Bei der Ausführung einer WebSphere MQ-JMS-Clientanwendung werden möglicherweise die folgenden Warnhinweise angezeigt:

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 WebSphere MQ-Services

Sie können die Funktionen eines Warteschlangenmanagers durch Benutzerexits, API-Exits oder installierbare Services erweitern. 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.

Information zum Schreiben und Kompilieren von Exits und installierbaren Services finden Sie im Abschnitt „Exits und installierbare Services schreiben und kompilieren” auf Seite 399.

Zugehörige Konzepte
Kanalexitprogramme für MQI-Kanäle
Zugehörige Verweise
API-Exitreferenz
Referenzinformationen zu installierbaren Services

Exits und installierbare Services schreiben und kompilieren

Sie können Exits schreiben und kompilieren, ohne eine Verbindung zu IBM WebSphere MQ -Bibliotheken unter UNIX, Linux und Windows herzustellen.

Informationen zu diesem Vorgang

Windows Linux UNIX Dieser Abschnitt gilt nur für Windows-, UNIX and Linux -Systeme. Informationen zum Schreiben von Exits und installierbaren Services für andere Plattformen finden Sie in den relevanten plattformspezifischen Abschnitten.

Wenn IBM WebSphere MQ nicht in einem Standardverzeichnis installiert ist, müssen Sie Ihre Exits ohne Verbindung zu irgendwelchen IBM WebSphere MQ-Bibliotheken schreiben und kompilieren.

Sie können Exits auf Windows- und UNIX and Linux-Systemen ohne Verbindung zu den folgenden IBM WebSphere 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 WebSphere 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 zum Schreiben von API-Exits finden Sie im Abschnitt „API-Exits schreiben” auf Seite 414.
   - 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 zum Schreiben von Kanalexits finden Sie in „Kanalexitprogramme schreiben” auf Seite 425.

- 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 hierzu finden Sie in den Abschnitten „Datenkonvertierungsexit für WebSphere MQ for Windows schreiben" auf Seite 447 und „Datenkonvertierungsexit für WebSphere MQ auf Systemen mit UNIX and Linux schreiben" auf Seite 444.
  - 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 zum Schreiben von Datenkonvertierungsexits finden Sie im Abschnitt „Datenkonvertierungsexits schreiben" auf Seite 442.
- 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 zum Schreiben von Pre-Connect-Exits finden Sie im Abschnitt „Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren" auf Seite 450.
- 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 zum Schreiben von Veröffentlichungsexits finden Sie in „Veröffentlichungsexits schreiben und kompilieren" auf Seite 452.
- 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 in „Exits für Clusterauslastung schreiben und kompilieren" auf Seite 454.

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 „WebSphere MQ-Beispielprogramme" auf Seite 100.

3. Kompilieren Sie den Exit:
   - Stellen Sie keine Verbindung zu den IBM WebSphere MQ-Bibliotheken her.
   - Nehmen Sie keinen integrierten RPath in IBM WebSphere 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 416.
  - Kanalexits, Veröffentlichungsexits, Exits für Clusterauslastung: „Kanalexitprogramme auf Windows-, UNIX and Linux -Systemen kompilieren” auf Seite 441.
  - Datenkonvertierungsexits: „Datenkonvertierungsexits schreiben” auf Seite 442.

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. Unter Windows und Linuxkönnen Sie den Pfad mit dem WebSphere 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 ein Exit sowohl in einem bestimmten Installationsverzeichnis als auch im Standardpfadverzeichnis positioniert wird, verwendet die im Pfad genannte Installation von WebSphere MQ den Exit im bestimmten Installationsverzeichnis. Der Exit wird beispielsweise in /exits/installation2 und /exits, jedoch nicht in /exits/installation1 gespeichert. Die WebSphere MQ-Installation installation2 verwendet den Exit aus dem Verzeichnis /exits/installation2. Die WebSphere MQ-Installation installation1 verwendet den Exit aus dem Verzeichnis /exits.
5. Konfigurieren Sie den Exit, falls erforderlich:
   - Installierbare Services: „Services und Komponenten konfigurieren” auf Seite 410
   - API-Exits: „API-Exits konfigurieren” auf Seite 419.
   - Kanalexits: „Kanalexits konfigurieren” auf Seite 441
   - Veröffentlichungsexits: „Veröffentlichungsexits konfigurieren” auf Seite 453
   - Pre-Connect-Exits: „Zeilengruppe für PreConnect der Clientkonfigurationsdatei” auf Seite 451

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.

Für die Bereitstellung von installierbaren WebSphere MQ-Services sprechen vor allem folgende Gründe:

- Um Ihnen die Flexibilität zu bieten, selbst auszuwählen, ob Sie die von den WebSphere MQ-Produkten bereitgestellten Komponenten verwenden oder diese Komponenten durch andere Komponenten ersetzen oder erweitern wollen.
- Um Anbietern die Möglichkeit zu bieten, mit ihren eigenen Komponenten neue Technologien bereitzustellen, ohne die WebSphere MQ-Produkte intern ändern zu müssen.
- Um WebSphere MQ die schnellere und kostengünstigere Nutzung neuer Technologien zu ermöglichen und dadurch Produkte früher und zu niedrigeren Preisen anbieten zu können.

Installierbare Services und *Servicekomponenten* sind Bestandteil der Produktstruktur von WebSphere 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 WebSphere MQ bereitgestellten Produkte installierbare Komponenten bereitstellen. In Tabelle 52 auf Seite 402 sehen Sie eine Zusammenfassung der installierbaren Services und ihrer Komponenten.

Tabelle 52. 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 **Anmerkung:** Für gemeinsam genutzte Warteschlangen muss das Attribut *Scope* auf CELL gesetzt sein.	• 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.

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 WebSphere MQ for AIX beispielsweise führt die bereitgestellte Berechtigungsservicekomponente, der Objektberechtigungsmanager, alle verfügbaren Funktionen aus. Weitere Informationen hierzu finden Sie unter „Schnittstelle für Berechtigungsservice” auf Seite 407. 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 72 auf Seite 403 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.

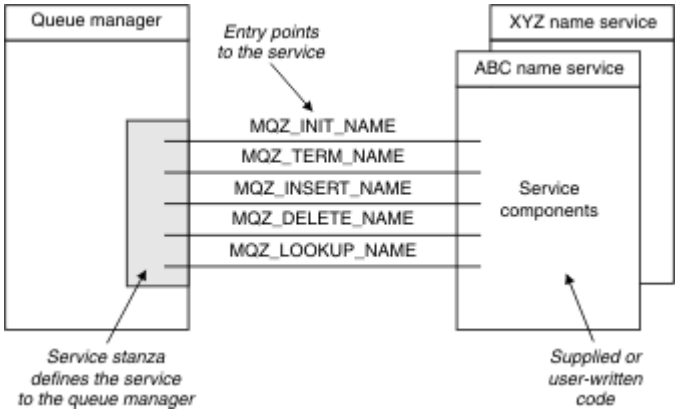


Abbildung 72. Zusammenhang zwischen Services, Komponenten und Eingangspunkten

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 Registrierungsschlüssels 'Service' von ServiceComponentdem Wert des Registrierungsschlüssels 'Name' entsprechen und wie folgt definiert sein: HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\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. Da unter Windows die Windows-Registrierungsdatenbank verwendet wird, gibt WebSphere MQ einen Aufruf **RegEnumKey** aus, der die Werte in alphabetischer Reihenfolge zurückgibt. Somit werden die Services unter Windows in alphabetischer Reihenfolge aufgerufen, so wie sie in der Registrierungsdatenbank definiert sind.

Die Anordnung der ServiceComponent-Definitionen ist wichtig. Diese Anordnung legt fest, in welcher Reihenfolge Komponenten für einen bestimmten Service ausgeführt werden. Beispiel: Der Berechtigungsservice (AuthorizationService) unter Windows ist mit dem Standardobjektberechtigungsmanager 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)
Die Berechtigungsservicekomponente, die zusammen mit den WebSphere MQ-Produkten bereitgestellt wird, wird als Objektberechtigungsmanager (Object Authority Manager, OAM) bezeichnet.

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 Steuerbefehle beschrieben.

Der Objektberechtigungsmanager arbeitet mit der *Entität* eines Principals oder einer Gruppe.

- Auf UNIX and Linux-Systemen:
  - Der Prinzipal ist eine Benutzer-ID oder eine ID, die einem Anwendungsprogramm zugeordnet ist, das für einen Benutzer ausgeführt wird.
  - Die Gruppe ist eine systemdefinierte Zusammenstellung von Prinzipalen für UNIX oder Linux.
  - 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.
- Auf Windows-Systemen:
  - Der Prinzipal ist eine Windows-Benutzer-ID oder eine ID, die einem Anwendungsprogramm zugeordnet ist, das für einen Benutzer ausgeführt wird.
  - Die Gruppe ist eine Windows-Gruppe.
  - Berechtigungen können auf Principal- oder Gruppenebene erteilt oder entzogen werden.

Wenn eine MQI-Anforderung gestellt oder ein Befehl ausgegeben wird, prüft der Objektberechtigungsmanager die Berechtigung der Entität, die der jeweiligen Operation zugeordnet ist, und prüft, ob sie zu folgenden Aktionen berechtigt ist:

- 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.

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: Dazu ***muss*** das Attribut *Scope* der fernen Warteschlange 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) gehört zum WebSphere MQ-Framework.

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.

Zusätzlich stellt der Berechtigungsservice in WebSphere MQ for Windows folgende Eingangspunkte für die Verwendung durch 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
Ermittelt den Warteschlangenmanagernamen für die angegebene Warteschlange.

MQZ_INSERT_NAME
Fügt einen Eintrag mit dem Namen des Warteschlangenmanagers, dem die angegebene Warteschlange zugeordnet ist, in dem vom Service verwendeten 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 53 auf Seite 409 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 53 auf Seite 409). 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 53. 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

Verwenden Sie für die Konfiguration von Servicekomponenten die Konfigurationsdateien des Warteschlangenmanagers. Eine Ausnahme bilden hierbei Windows-Systeme, auf denen es für jeden Warteschlangenmanager eine eigene Zeilengruppe in der Registrierungsdatenbank gibt.

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 410 und „Format der Zeilengruppe 'ServiceComponent'" auf Seite 411.

   Die Berechtigungsservicekomponente, der Objektberechtigungsmanager oder OAM, wird im Rahmen des Produkts bereitgestellt. Beim Erstellen eines Warteschlangenmanagers wird die Konfigurationsdatei des Warteschlangenmanagers (bzw. die Registrierungsdatenbank auf Windows-Systemen) automatisch aktualisiert, sodass sie die korrekten Zeilengruppen für den Berechtigungsservice und die Standardkomponente (Objektberechtigungsmanager) enthält. 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>
```

Dabei gilt:

<service_name>
Der Name des Service. Dieser wird durch den Service definiert.

<entries>
Die Anzahl der für den Service definierten Eingangspunkte. Dazu gehören die Initialisierungs-und Beendigungseingangspunkte

Format der Zeilengruppe 'Service' für Windows-Systeme

Auf Windows-Systemen umfasst die Zeilengruppe *Service* ein Attribut SecurityPolicy.

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 für den Service definierten Eingangspunkte. Dazu gehören die Initialisierungs-und Beendigungseingangspunkte

<policy>
`NTSIDsRequired` (Windows-Sicherheitskennung) 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 konfigurieren: Windows-Systeme” auf Seite 412.

Format der Zeilengruppe 'ServiceComponent'

Die Zeilengruppe 'ServiceComponent' hat folgendes Format:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

Dabei gilt:

<service_name>
Der Name des Service. Dieser muss dem in einer 'Service'-Zeilengruppe angegebenen Wert für `Name` entsprechen.

<component_name>
Der beschreibende Name der Servicekomponente. Dieser Name muss eindeutig sein und darf nur Zeichen enthalten, die für die Namen von WebSphere MQ-Objekten (beispielsweise Warteschlangennamen) zulässig 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.

<module_name>
Der Name des Moduls, das den Code für diese Komponente enthält.

<size>
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.

Zeilengruppen für Berechtigungsservice konfigurieren: UNIX and Linux-Systeme
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 WebSphere MQ for AIX zwei Berechtigungsservicekomponenten. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere 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 73. 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.

Zeilengruppen für Berechtigungsservice konfigurieren: Windows-Systeme
Unter WebSphere MQ for Windows verfügt jeder Warteschlangenmanager in der Registrierungsdatenbank über eine 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 WebSphere MQ -Services hinzufügen. Das Attribut `SsecurityPolicy` hat nur dann eine Wirkung, wenn der in der Zeilengruppe *Service* angegebene Service der Berechtigungsservice (d. h. der Standard-Objektberechtigungsmanager (OAM)) ist. 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 für eine bestimmte Benutzer-ID keine Windows-Sicherheits-ID (NT SID) an den OAM übergeben wurde, wird versucht, die entsprechende SID durch Durchsuchen der relevanten Sicherheitsdatenbanken zu erhalten.

`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' für Windows-Systeme” auf Seite 410. Weitere allgemeine Informationen zur Sicherheit finden Sie unter Sicherheit unter Windows, UNIX and Linux 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.

Zeilengruppen für den Namensservice konfigurieren: UNIX- und Linux-Systeme
Geben Sie hier Ihre Kurzbeschreibung an; sie wird für den ersten Absatz verwendet und ist abstrakt.

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 74. Zeilengruppen für den Namensservice aus der Datei 'qm.ini' (für UNIX and Linux-Systeme)

Anmerkung: Auf Windows-Systemen werden die Zeilengruppeninformationen für den Namensservice in der Registrierungsdatenbank gespeichert.

OAM nach Änderung einer Benutzerberechtigung aktualisieren
In WebSphere MQ können Sie die Berechtigungsgruppeninformationen des OAM sofort nach der Änderung der Zugehörigkeit eines Benutzers zu einer Berechtigungsgruppe aktualisieren und so Änderungen auf Betriebssystemebene übernehmen, ohne den Warteschlangenmanager stoppen und neu starten zu müssen. Dazu geben Sie den Befehl **REFRESH SECURITY** aus.

Anmerkung: Mit dem Befehl `setmqaut` geänderte Berechtigungen werden vom OAM sofort implementiert.

Warteschlangenmanager speichern Berechtigungsdaten in der lokalen Warteschlange SYSTEM.AUTH.DATA.QUEUE. Diese Daten werden von `amqzfuma.exe` verwaltet.

Zugehörige Verweise
REFRESH SECURITY

API-Exits schreiben und kompilieren

Mit API-Exits können Sie Code schreiben, mit dem das Verhalten von WebSphere MQ-API-Aufrufen wie MQPUT und MQGET geändert werden kann, und diesen Code dann unmittelbar vor oder hinter diesen Aufrufen einfügen.

Anmerkung: Wird für WebSphere MQ for z/OS nicht unterstützt.

Gründe für die Verwendung von API-Exits

Jede Ihrer Anwendungen erfüllt eine bestimmte Aufgabe und der Anwendungscode sollte diese Aufgabe so effizient wie möglich ausführen. Auf einer höheren Ebene kann es sinnvoll sein, für **alle** Anwendungen, die einen bestimmten Warteschlangenmanager verwenden, Standards oder Geschäftsprozesse 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 WebSphere MQ angegeben haben, ruft der Warteschlangenmanager Ihren Exitcode an den registrierten Punkten automatisch auf.

Die auszuführenden API-Exitroutinen sind in Zeilengruppen auf IBM i-, Windows-und UNIX and Linux -Systemen 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 das gesamte WebSphere MQ-Produkt angegeben, die beim Start der Warteschlangenmanager angewendet werden. Diese können durch für einzelne Warteschlangenmanager definierte Routinen überschrieben werden (siehe Artikel „3” auf Seite 414 in dieser Liste).
2. In 'ApiExitTemplate' in der Datei 'mqs.ini' werden Routinen für das gesamte WebSphere MQ-Produkt angegeben, die bei der Erstellung eines neuen Warteschlangenmanagers in die Gruppe 'ApiExitLocal' kopiert wurden (siehe Artikel „3” auf Seite 414 in dieser Liste).
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. In dem in „API-Exits konfigurieren” auf Seite 419 beschriebenen Attribut Sequence ist die Reihenfolge festgelegt, in der die in den Zeilengruppen definierten Routinen ausgeführt werden.

API-Exits in mehreren Installationen von WebSphere MQ verwenden

Vergewissern Sie sich, dass die API-Exits, die für die frühere Version von WebSphere MQ geschrieben wurden, mit allen Versionen verwendet werden können, weil die Änderungen, die an Exits in Version 7.1 vorgenommen wurden, möglicherweise mit einer früheren Version nicht fehlerfrei arbeiten. Weitere Informationen zu den an den Exits vorgenommenen Änderungen finden Sie in „Exits und installierbare Services schreiben und kompilieren” auf Seite 399.

Die für die API-Exits amqsaem und amqsaxe bereitgestellten Beispiele spiegeln die Änderungen wider, die beim Schreiben von Exits erforderlich sind. Die Clientanwendung muss sicherstellen, dass die korrekten WebSphere MQ-Bibliotheken, die der Installation des der Anwendung zugeordneten Warteschlangenmanagers entsprechen, mit dieser Anwendung verknüpft werden, bevor sie gestartet wird.

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 WebSphere MQ-Ressource für nachfolgenden Zugriff

- MQCLOSE zum Schließen einer WebSphere 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 zum Abrufen der Attribute einer WebSphere 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 119.

Weitere Informationen zu API-Exit-Aufrufen, externen Steuerblöcken und zugehörigen Themen finden Sie im Abschnitt API exit reference.

Allgemeine Informationen zum Schreiben, Kompilieren und Konfigurieren eines Exits finden Sie im Abschnitt „Exits und installierbare Services schreiben und kompilieren” auf Seite 399.

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 wird auf einen Wert gesetzt, der von WebSphere MQ generiert wird und für eine Verbindung spezifisch ist. 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 werden die Eigenschaften in 'ExitMsgHandle' gelöscht und durch die Eigenschaften ersetzt, die bei der Registrierung des MQ_INIT_EXIT im Feld 'ExitProperties' angegeben wurden (außer Nachrichtendeskriptoreigenschaften). Diese Eigenschaften werden der abrufenden Anwendung nicht zur Verfügung gestellt. Wenn die abrufende Anwendung eine Nachrichtenkennung im Feld 'MQGMO (Get message options)' angegeben hat, sind alle Eigenschaften, die dieser Kennung zugeordnet sind, einschließlich der Nachrichtendeskriptoreigenschaften, für den API-Exit verfügbar. Um zu verhindern, dass 'ExitMsgHandle' mit Eigenschaften gefüllt wird, können Sie MQHM_NONE angeben.

Ein Musterprogramm (amqsaem0.c) wird bereitgestellt, um die Verwendung von Nachrichtenkennungen in API-Exits zu veranschaulichen.

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 119 beschriebene Musterprogramm verwendet werden. Für andere Plattformen als Windows-Systeme finden Sie den API-Exit-Mustercode im Verzeichnis *MQ_INSTALLATION_PATH*/samp und die kompilierte und verbundene gemeinsam genutzte Bibliothek im Verzeichnis *MQ_INSTALLATION_PATH*/samp/bin. Für Windows-Systeme finden Sie den API-Exit-Mustercode im Verzeichnis *MQ_INSTALLATION_PATH*\Tools\c\Samples.*MQ_INSTALLATION_PATH* Das Verzeichnis, in dem WebSphere MQ installiert wurde.

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 54. 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

API-Exits auf Unix- und Linux-Systemen kompilieren
Beispiele für die Kompilierung von API-Exits auf UNIX-und Linux -Systemen.

Auf allen Plattformen ist der Eingangspunkt in das Modul 'MQStart'.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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 -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
     amqsaxe0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
 xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
     amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Auf HP-UX-Itanium-Plattform

32-Bit-Anwendungen

Nicht-thread-basiert

Kompilieren Sie den API-Exit-Quellcode:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_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 \
     -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

32-Bit-Anwendungen

Nicht-thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Nicht-thread-basiert

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
     -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Auf Windows-Systemen
API-Beispielexitprogramm `amqsaxe0.c`unter Windows kompilieren und verlinken

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 Beispielen in Abbildung 75 auf Seite 419 oder Abbildung 76 auf Seite 419 an, um `amqsaxe0.c` unter Windows zu kompilieren und zu verknüpfen. Die Befehle funktionieren unter Microsoft Visual Studio 2005, 2008 oder 2010. In den Beispielen wird vorausgesetzt, dass das Verzeichnis WebSphere MQ `C:\Program Files\IBM\WebSphere 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 75. amqsaxe0.c unter 32-Bit-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 76. amqsaxe0.c unter 64-Bit-Windows kompilieren und verbinden

Zugehörige Konzepte
„API Exit-Beispielprogramm” auf Seite 119
Das API Exit-Beispielprogramm generiert einen MQI-Trace zu einer benutzerdefinierten Datei mit einem in der Umgebungsvariable MQAPI_TRACE_LOGFILE definierten Präfix.

API-Exits konfigurieren
Durch eine Änderung der Konfigurationsdaten kann IBM WebSphere 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 WebSphere 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, 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 und im Work-Path , der 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
```

Unter Windows:

```
C:\Program Files\IBM\WebSphere 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 WebSphere MQ Explorer auf der IBM WebSphere 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 WebSphere MQ Explorer auf der IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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. Die übergebenen Daten sind 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

Diese Themensammlung enthält Informationen zu WebSphere 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 442.

Ein Teil dieser Informationen gilt auch für Exits auf MQI-Kanälen, mit denen MQI-Clients von WebSphere MQ mit Warteschlangenmanagern verbunden werden. 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 55 auf Seite 422 zeigt die Kanalexitarten, die für die einzelnen Kanaltypen verfügbar sind.

Tabelle 55. 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

Tabelle 55. Für die einzelnen Kanaltypen verfügbare Kanalexits (Forts.)						
Kanaltyp	**Nachrichtenexit**	**Nachrichtenwiederholungsexit**	**Empfangsexit**	**Sicherheitsexit**	**Sendeexit**	**Autodefinitionsexit**
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

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), wie im Abschnitt Definitionstabelle für Clientkanal 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 77 auf Seite 423 dargestellt.

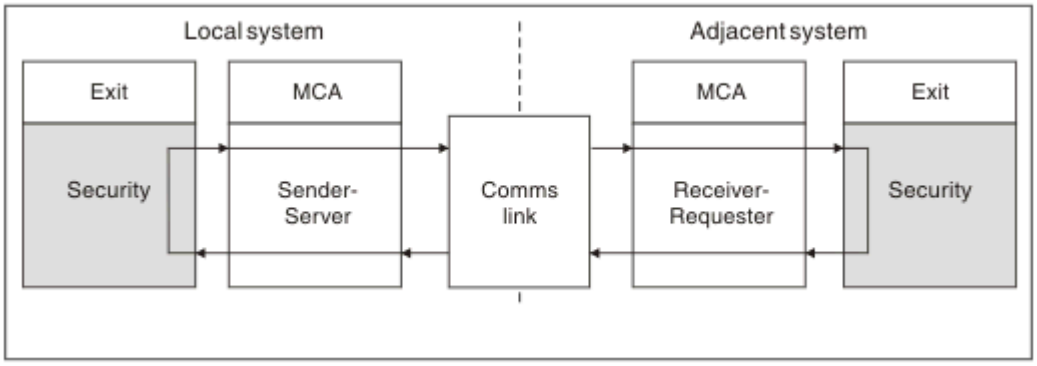


Abbildung 77. Sicherheitsexitschleife

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 78 auf Seite 424 dargestellt an den empfangenden Nachrichtenkanalagenten.

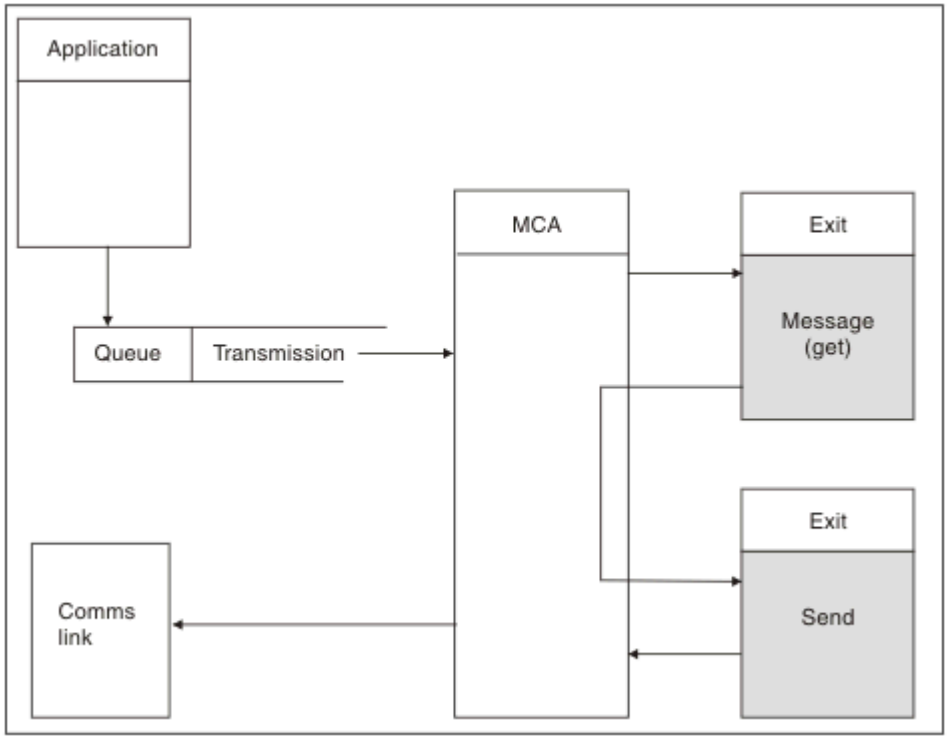


Abbildung 78. Beispiel eines Sendeexits auf der Sendeseite des Nachrichtenkanals

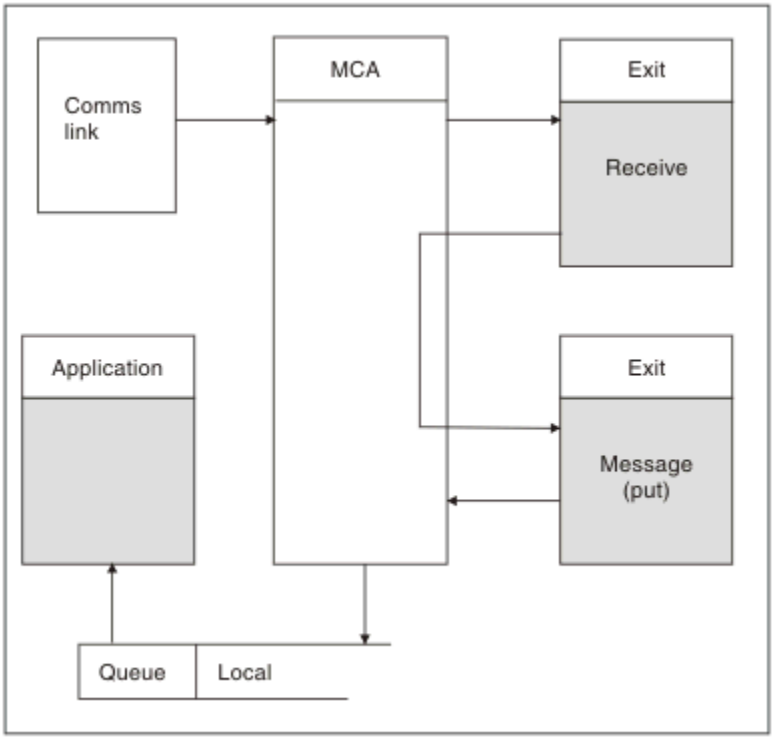


Abbildung 79. Beispiel eines Empfangsexits auf der Empfangsseite des Nachrichtenkanals

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 79 auf Seite 424 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. Wurde der Exit mit MQM.LIB) verbunden und geben Sie beim MQCONN-Aufruf keinen Warteschlangenmanagernamen an, versucht der Exit, sich mit dem Standardwarteschlangenmanager auf Ihrem System zu verbinden. Wurde der Exit mit MQM.LIB verbunden und geben Sie den Namen des Warteschlangenmanagers an, der über das MQCD-Feld 'QMgrName' an den Exit übergeben wurde, versucht der Exit, sich mit diesem Warteschlangenmanager zu verbinden. 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 WebSphere 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 WebSphere 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.

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:
  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 422 finden Sie Details zum Aufruf.

Kanalexitprogramme unter Windows, UNIX and Linux schreiben
Dieser Abschnitt enthält Informationen zum Erstellen von Kanalexitprogrammen für Windows-Systeme sowie für Systeme mit UNIX and Linux.

Folgen Sie den unter „Exits und installierbare Services schreiben und kompilieren” auf Seite 399 beschriebenen Anweisungen. Berücksichtigen Sie ggf. die folgenden kanalexitspezifischen Informationen:

Der Exit muss in C geschrieben werden und ist unter Windows eine DLL.

Definieren Sie im Exit eine 'MQStart()'-Dummyroutine und geben Sie `MQStart` in der Bibliothek als Eingangspunkt an. In Abbildung 80 auf Seite 426 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 80. 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 81 auf Seite 427. Weitere Informationen zum Schreiben von Kanalexitprogrammen finden Sie im Abschnitt „Kanalexitprogramme schreiben” auf Seite 425.

```
EXPORTS
ChannelExit
```

Abbildung 81. DEF-Musterdatei für Windows

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 WebSphere MQ-Benutzer-IDs zugeordnet werden sollen. Die SSL- und TLS-Unterstützung wird auch durch IBM WebSphere MQ bereitgestellt, um die Benutzer zu authentifizieren und eine Verschlüsselung und Dateninteгritätsprüfungen für Ihre Daten zu ermöglichen. Weitere Informationen zu SSL und TLS finden Sie im Abschnitt zur WebSphere MQ-Unterstützung für SSL und TLS. 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 Version 7.1 geschrieben wurden, ist zu beachten, dass in früheren Versionen von IBM WebSphere 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 Version 7.1 unterstützt nun eine Reihe neuer Sicherheitsattribute. Um auf diese Attribute zuzugreifen, ruft IBM WebSphere MQ Version 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 Version 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 Version 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 WebSphere MQ-Peernamensfilter, die über das Feld SSLPEER einer Kanaldefinition angegeben werden, sind nicht betroffen und können weiterhin auf dieselbe Weise wie in früheren Releases verwendet werden. Dies liegt daran, dass der Abgleichsalgorithmus von WebSphere MQ für Peernamen aktualisiert wurde, sodass vorhandene SSLPEER-Filter verarbeitet werden, ohne dass die Kanaldefinitionen geändert werden müssen. 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 82 auf Seite 429bis Abbildung 85 auf Seite 431 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 82 auf Seite 429:

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 82. 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 83. 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 84. 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 85. 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. 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. Auf einem SSL-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 Einschränkung des Zugriffs über den SSL-Kanal.
   - Zur Änderung von MQCD.MCAUserIdentifier auf Basis des Namens.

Zugehörige Konzepte

Kanalauthentifizierungsdatensätze

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts

Sicherheitsexit schreiben

Sie können den Sicherheitsexit-Entwurfscode zum Schreiben von Sicherheitsexits verwenden.

Abbildung 86 auf Seite 432 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 86. Sicherheitsexit-Entwurfscode

Der WebSphere MQ-Standardeingangspunkt 'MQStart' muss vorhanden sein, das Ausführen von Funktionen ist jedoch nicht erforderlich. Die Änderung des Funktionsnamens ('EntryPoint' im vorliegenden Beispiel) ist möglich, die Funktion muss jedoch bei der Kompilierung und Verbindung der Bibliothek exportiert werden. Wie im vorherigen Beispiel muss 'pChannelExitParms' auf PMQCXP und 'pChannelDefinition' auf PMQCD verweisen. Allgemeine Informationen zum Aufruf von Kanalexits und der Verwendung von Parametern finden Sie im Abschnitt MQ_CHANNEL_EXIT. Diese Parameter werden wie folgt in einem Sicherheitsexit verwendet:

PMQVOID pChannelExitParms
Eingabe/Ausgabe

Zeiger zur MQCXP-Struktur - Verweis auf PMQCXP zum Zugriff auf Felder. Diese Struktur wird zur Kommunikation zwischen dem Exit und dem Nachrichtenkanalagenten verwendet. Die folgenden Felder in MQCXP sind in Bezug auf Sicherheitsexits von besonderem Interesse:

ExitReason
Informiert den Sicherheitsexit über den aktuellen Status des Sicherheitsaustauschs und wird verwendet, wenn über die zu ergreifende Maßnahme entschieden werden soll.

ExitResponse
Die Antwort auf den Nachrichtenkanalagenten, die den nächsten Schritt im Sicherheitsaustausch festlegt.

ExitResponse2
Zusätzliche Steuermarkierungen, die regeln, wie der Nachrichtenkanalagent die Antwort des Sicherheitsexits interpretiert.

ExitUserArea
16 Byte Speicher (maximal), der vom Sicherheitsexit zur Beibehaltung des Status zwischen Anrufen verwendet werden kann.

ExitData
Enthält die im Feld SCYDATA der Kanaldefinition angegebenen Daten (32 Bytes, die rechts durch Leerzeichen aufgefüllt sind).

PMQVOID pChannelDefinition
Eingabe/Ausgabe

Zeiger zur MQCD-Struktur - Verweis auf PMQCD zum Zugriff auf Felder. Dieser Parameter enthält die Kanaldefinition. Die folgenden Felder in der MQCD sind in Bezug auf Sicherheitsexits von besonderem Interesse:

ChannelName
Der Kanalname (20 Bytes, die rechts durch Leerzeichen aufgefüllt sind).

ChannelType
Ein Code zur Definition des Kanaltyps.

MCA User Identifier
Diese aus drei Feldern bestehende Gruppe wird mit dem Wert des in der Kanaldefinition angegebenen MCAUSER-Felds initialisiert. Durch den Sicherheitsexit in diesen Feldern angegebene Benutzer-IDs werden zur Zugriffssteuerung verwendet (nicht zutreffend für SDR-, SVR-, CLNTCONN- oder CLUSSDR-Kanäle).

MCAUserIdentifier
Die ersten 12 Bytes der Kennung, die rechts durch Leerzeichnen aufgefüllt sind.

LongMCAUserIdPtr
Zeiger zu einem Puffer, der die Kennung in gesamter Länge enthält (Nullabschluss nicht garantiert). Hat Vorrang vor 'MCAUserIdentifier'.

LongMCAUserIdLength
Zeichenfolge, auf die durch 'LongMCAUserIdPtr' verwiesen wird - muss festgelegt werden, wenn 'LongMCAUserIdPtr' festgelegt ist.

Remote User Identifier
Findet nur bei CLNTCONN/SVRCONN-Kanalpaaren Anwendung. Wenn kein CLNTCONN-Sicherheitsexit definiert ist, werden diese drei Felder durch den Client-Nachrichtenkanalagenten initialisiert und können eine Benutzer-ID aus der Clientumgebung enthalten, die durch einen SVRCONN-Sicherheitsexit zur Authentifizierung oder zur Angabe der Benutzer-ID des Nachrichtenkanalagenten verwendet werden kann. Wenn ein CLNTCONN-Sicherheitexit definiert ist, werden diese drei Felder nicht initialisiert und können durch den CLNTCONN-Sicherheitsexit festgelegt werden bzw. können Sicherheitsnachrichten dazu verwendet werden, eine Benutzer-ID vom Client zum Server zu senden.

Remote User Identifier
Die ersten 12 Bytes der Kennung, die rechts durch Leerzeichnen aufgefüllt sind.

LongRemoteUserIdPtr
Zeiger zu einem Puffer, der die Kennung in gesamter Länge enthält (Nullabschluss nicht garantiert). Hat Vorrang vor 'RemoteUserIdentifier'.

LongRemoteUserIdLength
Zeichenfolge, auf die durch 'LongRemoteUserIdPtr' verwiesen wird - muss festgelegt werden, wenn 'LongRemoteUserIdPtr' festgelegt ist.

PMQLONG pDataLength
Eingabe/Ausgabe

Zeiger zu MQLONG. Enthält die Länge der beim Aufruf des Sicherheitsexits in 'AgentBuffer' enthaltenen Sicherheitsexits. Muss durch einen Sicherheitsexit auf die Länge einer in 'AgentBuffer' oder 'ExitBuffer' gesendeten Nachricht festgelegt werden.

PMQLONG pAgentBufferLength
Eingabe

Zeiger zu MQLONG. Die Länge der beim Aufruf des Sicherheitsexits in 'AgentBuffer' enthaltenen Daten.

PMQVOID pAgentBuffer
Eingabe/Ausgabe

Beim Aufruf des Sicherheitsexits verweist dieser Parameter auf vom Partnerexit gesendete Nachrichten. Wenn bei 'ExitResponse2' in der MQCXP-Struktur das Flag MQXR2_USE_AGENT_BUFFER gesetzt ist (Standardeinstellung), muss dieser Parameter durch einen Sicherheitsexit so gesetzt werden, dass er auf die gesendeten Nachrichtendaten verweist.

PMQLONG pExitBufferLength
Eingabe/Ausgabe

Zeiger zu MQLONG. Dieser Parameter wird beim ersten Aufruf eines Sicherheitsexits auf 0 gesetzt und der zurückgegebene Wert wird zwischen Aufrufen des Sicherheitsexits während eines Sicherheitsaustauschs beibehalten.

PMQPTR pExitBufferAddr
Eingabe/Ausgabe

Dieser Parameter wird beim ersten Aufruf eines Sicherheitsexits auf einen Nullzeiger gesetzt und der zurückgegebene Wert wird zwischen Aufrufen des Sicherheitsexits während eines Sicherheitsaustauschs beibehalten. Wenn das Flag MQXR2_USE_EXIT_BUFFER in 'ExitResponse2' in der MQCXP-Structure gesetzt ist, muss dieser Parameter durch einen Sicherheitsexit so gesetzt werden, dass er auf die gesendeten Nachrichtendaten verweist.

Unterschiede im Verhalten von auf CLNTCONN/SVRCONN-Kanalpaaren und anderen Kanalpaaren definierten Sicherheitsexits
Sicherheitsexits können auf allen Kanaltypen definiert werden. Allerdings unterscheidet sich das Verhalten von auf CLNTCONN/SVRCONN-Kanalpaaren definierten Sicherheitsexits geringfügig von dem von Sicherheitsexits, die auf anderen Kanalpaaren definiert wurden.

Ein Sicherheitsexit auf einem CLNTCONN-Kanal kann die Remotebenutzer-ID in der Kanaldefinition so festlegen, dass die Verarbeitung durch einen SVRCONN-Partnerexit durchgeführt wird bzw. eine OAM-Autorisierung erfolgt, wenn kein SVRCONN-Sicherheitsexit definiert und das MCAUSER-Feld von SVRCONN nicht festgelegt ist.

Wenn kein CLNTCONN-Sicherheitsexit definiert ist, wird die Remotebenutzer-ID durch den Client-Nachrichtenkanalagenten in der Kanaldefinition auf eine Benutzer-ID aus der Clientumgebung gesetzt (die leer sein kann).

Ein Sicherheitsaustausch zwischen Sicherheitsexits auf einem CLNTCONN-und-SVRCONN-Kanalpaar wird erfolgreich abgeschlossen, wenn der SVRCONN-Sicherheitsexit als 'ExitResponse' MQXCC_OK zurückgibt. Ein Sicherheitsaustausch zwischen anderen Kanalpaaren wird erfolgreich abgeschlossen, wenn der Sicherheitsexit, der den Austausch initiiert hat, als 'ExitResponse' MQXCC_OK zurückgibt.

Mit dem 'ExitResponse'-Code MQXCC_SEND_AND_REQUEST_SEC_MSG kann jedoch die Fortsetzung des Sicherheitsaustauschs erzwungen werden: Wenn im Feld 'ExitResponse' durch den Sicherheitsexit CLNTCONN oder SVRCONN MQXCC_SEND_AND_REQUEST_SEC_MSG zurückgegeben wird, muss der Partnerexit durch das Senden einer Sicherheitsnachricht (nicht MQXCC_OK oder eine Nullantwort) reagieren, um die Beendigung des Kanals zu verhindern. Bei auf anderen Kanälen definierten Sicherheitsexits resultiert die Rückgabe von MQXCC_OK im Feld 'ExitResponse' auf die Anfrage MQXCC_SEND_AND_REQUEST_SEC_MSG des Partnersicherheitsexits nicht in der Beendigung des Kanals, sondern in der Fortsetzung des Sicherheitsaustauschs, als wäre eine Nullantwort zurückgegeben worden.

SSPI-Sicherheitsexit
WebSphere MQ for Windows stellt einen Sicherheitsexit bereit, über den die Authentifizierung für WebSphere MQ-Kanäle mithilfe von SSPI (Security Services Programming Interface) möglich ist. Die SSPI stellt die integrierten Sicherheitsfunktionen von Windows bereit.

Dieser Sicherheitsexit kann auf dem WebSphere MQ-Client und dem WebSphere MQ-Server eingesetzt werden.

Die Sicherheitspakete werden aus der Datei 'security.dll' oder 'secur32.dll' geladen. Diese DLLs werden mit Ihrem Betriebssystem geliefert.

Die unidirektionale Authentifizierung wird unter Windows über NTLM-Authentifizierungsservices bereitgestellt. Die Zweiwegeauthentifizierung wird unter Windows 2000 über Kerberos-Authentifizierungsservices 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 „Verwenden des SSPI-Sicherheitsexits auf Windows-Systemen” auf Seite 179

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 des Speicherbereichs im Übertragungspuffer für Nachrichtendaten freigehalten werden. Die Kanalleistung kann beeinträchtigt werden, wenn reservierter Speicherplatz für große Datenmengen verwendet wird.

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. WebSphere 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.

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.) 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 282 und „Referenznachrichtenbeispiel ausführen” auf Seite 149.

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. Auf UNIX-Systemen funktioniert die automatische Konvertierung von Benutzer-IDs in Kleinbuchstaben nicht, wenn Sie aus einem bestimmten Grund einen Nachrichtenexit angeben. Siehe Security of objects on UNIX and Linux systems.
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. Aus diesem Grund 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.

   Daher ist es möglich, MQI-Aufrufe unter Synchronisationspunktsteuerung über ein Kanalnachrichtenexitprogramm auszugeben.

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 mit Formatnamen, die mit den drei Zeichen 'MQH' beginnen (z. B. "MQHRF ") die nicht anders erwähnt sind

Vorgehensweise bei der Verarbeitung von Headern

Der Parameter für die Formatangabe der einzelnen WebSphere 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 WebSphere 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. Informa-

tionen 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, aber keine Definition für diesen Kanal existiert (nicht für WebSphere 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 existiert. 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 gilt der Exit auch für WebSphere MQ for z/OS. 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. Ist kein Exit für die automatische Kanaldefinition angegeben, wird unter z/OS standardmäßig ein verteilter Exitname im Format *[path]/libraryname(function)* untersucht und es werden bis zu acht Zeichen der Funktion, falls vorhanden, oder des Bibliotheksnamen verwendet. Unter z/OS muss ein Exitprogramm für die automatische Kanaldefinition die Felder "MsgExitPtr", "MsgUserDataPtr", "SendExitPtr", "SendUserDataPtr", "ReceiveExitPtr" und "ReceiveUserDataPtr" und nicht die Felder "MsgExit", "MsgUserData", "SendExit", "SendUserData", "ReceiveExit" und "ReceiveUserData" selbst ändern.

Weitere Informationen finden Sie im Abschnitt zur automatischen Definition von Kanälen.

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

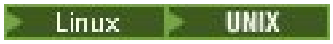
Mithilfe der folgenden Beispiele können Sie Kanalexitprogramme für Systeme unter Windows, UNIX and Linux kompilieren.

Windows

Der Compiler- und Linkerbefehl für Kanalexitprogramme lautet unter Windows wie folgt:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

UNIX-und Linux -Systeme



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.

Es folgt ein Beispiel für Compiler- und Linkerbefehle 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
```

Es folgt ein Beispiel für Compiler- und Linkerbefehle 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-Musterbefehle für Kanalexits auf Linux-Plattformen mit einem 32-Bit-Wartschlangenmanager:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Compiler- und Linker-Musterbefehle für Kanalexits auf Linux-Plattformen mit einem 64-Bit-Warteschlangenmanager:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Es folgt ein Beispiel für Compiler- und Linkerbefehle 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 WebSphere 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 beispielswei-

se 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 WebSphere 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 in MQSeries for VSE/ESA nicht unterstützt.

Wenn Sie einen Aufruf MQPUT ausgeben, erstellt die Anwendung den Nachrichtendeskriptor (MQMD) der Nachricht. Da WebSphere MQ den Inhalt des MQMD unabhängig davon verstehen muss, auf welcher Plattform dieser erstellt wird, nimmt das System automatisch eine Konvertierung vor.

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. In diesem Abschnitt finden Sie Informationen zu der Datenkonvertierungsexitfunktion, die WebSphere MQ für Anwendungsdaten vorsieht, die 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. Auf diese Weise ist gewährleistet, dass die Daten während der Übertragung von WebSphere MQ konvertiert werden. 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 zum Konvertieren von Anwendungsdaten zwischen verschiedenen Systemcodierungen und CCSIDs (IDs der codierten Zeichensätze) müssen der WebSphere MQ-Datenkonvertierungsschnittstelle entsprechen.

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 56. 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

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 versucht, die erste Nachricht abzurufen, die dieses *Format* verwendet, seit die Anwendung eine Verbindung zum Warteschlangenmanager hergestellt hat. 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.

Datenkonvertierungsexit für WebSphere MQ auf Systemen mit UNIX and Linux schreiben

In diesem Abschnitt finden Sie Informationen zu den Schritten, die beim Schreiben von Datenkonvertierungsexitprogrammen für WebSphere MQ auf Systemen unter UNIX and Linux berücksichtigt werden 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. Der Objektname darf nicht mehr als acht belegte Zeichen enthalten, da das Feld *Format* nur über eine Länge von acht Zeichen verfügt. 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 444 festgelegten Nachrichtenformatnamen umbenennen. Die Entwurfsquellendatei und ihre Kopie sind schreibgeschützt.

   Die Entwurfsquellendatei trägt die Bezeichnung 'amqsvfc0.c'.
5. Unter WebSphere MQ for AIX steht außerdem eine Entwurfsexportdatei mit der Bezeichnung 'amqsvfc.exp' zur Verfügung. Kopieren Sie diese Datei und benennen Sie sie dabei in MYFORMAT.EXP um.
6. Der Entwurf umfasst die Musterheaderdatei 'amqsvmha.h' im Verzeichnis '*MQ_INSTALLATION_PATH*/inc', wobei '*MQ_INSTALLATION_PATH*' für das übergeordnete Verzeichnis steht, in dem WebSphere 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 444 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 444 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 444 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 445.
9. Stellen Sie die Ausgabe in das Exitverzeichnis. Das Standardexitverzeichnis ist `/var/mqm/exits` (32-Bit-Systeme) bzw. `/var/mqm/exits64` (64-Bit-Systeme). Diese Verzeichnisse können in der Datei 'qm.ini' bzw. 'mqclient.ini' geändert werden. Dieser Pfad kann für jeden einzelnen Warteschlangenmanager definiert werden. Es wird nur in dem betreffenden Pfad bzw. den betreffenden Pfaden nach dem Exit gesucht.

Anmerkung:

1. Falls im Befehl `crtmqcvx` gepackte Strukturen verwendet werden, müssen alle WebSphere MQ-Anwendungen auf diese Weise kompiliert werden.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann als *einziger* 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 WebSphere 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 -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
    MYFORMAT.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
    MYFORMAT.c -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_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 \
      -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

32-Bit-Anwendungen

Nicht-thread-basiert

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

64-Bit-Anwendungen

Ohne Thread

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
      -IMQ_INSTALLATION_PATH/inc
```

Thread-basiert

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
      -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64  -lsocket -lnsl -ldl
```

Datenkonvertierungsexit für WebSphere MQ for Windows schreiben

In diesem Abschnitt finden Sie Informationen zu den erforderlichen Schritten beim Schreiben von Datenkonvertierungsexitprogrammen für WebSphere MQ for Windows.

Führen Sie folgende Schritte aus:

1. Benennen Sie Ihr Nachrichtenformat. Der Name muss ins Feld *Format* des MQMD passen. Der im Feld *Format* angegebene Name darf keine führenden Leerzeichen enthalten. Abschließende Leerzeichen werden ignoriert. Der Objektname darf nicht mehr als acht belegte Zeichen enthalten, da das Feld *Format* nur über eine Länge von acht Zeichen verfügt.

   Eine .DEF-Datei namens amqsvfcn.def wird auch im Beispielverzeichnis *MQ_INSTALLATION_PATH*\Tools\C\Samplesbereitgestellt. *MQ_INSTALLATION_PATH* Das Verzeichnis, in dem WebSphere 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 448 festgelegten Nachrichtenformatnamen umbenennen.

   amqsvfc0.c befindet sich in *MQ_INSTALLATION_PATH*\Tools\C\Samples , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem WebSphere MQ installiert ist. (Das Standardinstallationsverzeichnis ist `C:\Program Files\IBM\WebSphere MQ`.)

   Das Gerüst enthält eine Beispielheaderdatei amqsvmha.h im Verzeichnis *MQ_INSTALLATION_PATH*\Tools\C\include . Vergewissern Sie sich, dass der include-Pfad auf dieses Verzeichnis zeigt, damit diese Datei berücksichtigt wird.

   Die Datei 'amqsvmha.h' enthält Makros, die von dem mit dem Befehl CRTMQCVX generierten Code verwendet werden. Umfasst die zu konvertierende Struktur Zeichendaten, wird von diesen Makros MQXCNVC aufgerufen.
5. Suchen Sie in der Quellendatei nach den folgenden Kommentarboxes und fügen Sie wie hier beschrieben Code ein:
   a. Am Ende der Quellendatei beginnt eine Kommentarbox mit:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Fügen Sie hier das in Schritt „3” auf Seite 448 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 448 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 448 hinzugefügten Funktionen ein.
6. Erstellen Sie die folgende Befehlsdatei:

```
cl -I MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C

MYFORMAT.DEF
```

Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem WebSphere MQ installiert ist.

7. Rufen Sie die Befehlsdatei auf, um den Exit als DLL-Datei zu kompilieren.
8. Stellen Sie die Ausgabe in das Exitunterverzeichnis unter dem WebSphere 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\MQSeries\CurrentVersion\Configuration\ClientExit□
Path\
```

   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. Falls im Befehl CRTMQCVX gepackte Strukturen verwendet werden, müssen alle WebSphere MQ-Anwendungen auf diese Weise kompiliert werden.
2. Datenkonvertierungsexitprogramme müssen simultan verwendbar sein.
3. Von einem Datenkonvertierungsexit aus kann als *einziger* MQI-Aufruf MQXCNVC ausgegeben werden.

Exit- und Switchloaddateien unter Windows-Betriebssystemen

Die Prozesse des Warteschlangenmanagers für IBM WebSphere MQ for Windows Version 7.5 werden im 32-Bit-Modus ausgeführt. Dadurch muss bei der Verwendung von 64-Bit-Anwendungen für einige Typen der Exit-Dateien und der XA-Switch-Ladedateien ebenfalls eine 32-Bit-Version für die Verwendung durch den Warteschlangenmanager verfügbar sein. Wenn die 32-Bit-Version der Exit-Datei oder der XA-Switch-Ladedatei erforderlich, jedoch nicht verfügbar ist, schlägt der relevante API-Aufruf oder API-Befehl fehl.

In der Datei qm.ini file für *ExitPath* werden zwei Attribute unterstützt. Dies sind ExitsDefaultPath=*MQ_INSTALLATION_PATH*\exits und ExitsDefaultPath64=*MQ_INSTALLATION_PATH*\exits64. *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist. Durch die Verwendung wird sichergestellt, dass die entsprechende Bibliothek gefunden werden kann. Wenn eine Exitdatei in einem WebSphere 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-Steuerübergabeexit	32 Bit	32 Bit und 64 Bit
Datenkonvertierungsexit	32 Bit	64 Bit
Serverkanalexits (alle Typen)	32 Bit	32 Bit
Clientkanalexits (alle Typen)	32 Bit	64 Bit
Exit für installierbaren Service	32 Bit	32 Bit
Service-Tracemodul	32 Bit	32 Bit und 64 Bit
WLM-Exit des Clusters	32 Bit	32 Bit

Dateitypen	32-Bit-Anwendungen	64-Bit-Anwendungen
Publish/Subscribe-Routing-Exit	32 Bit	32 Bit
Switchloaddateien der Datenbank	32 Bit	32 Bit und 64 Bit
AX-Bibliotheken für den externen Transaktionsmanager	32 Bit	64 Bit

Verbindungsdefinitionen unter Verwendung eines Vorverbindungsexits aus einem Repository referenzieren

WebSphere MQ-MQI-Clients können so konfiguriert werden, dass mithilfe einer PreConnect-Exitbibliothek ein Repository nach Verbindungsdefinitionen durchsucht wird.

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 die Verwaltung verschiedener Clientanwendungen, die auf die CCDT-Datei verweisen, schwierig ist, besteht ein flexibles Konzept darin, die Clientdefinitionen in einem globalen Repository wie z. B. einem LDAP-Verzeichnis, einer WebSphere-Registry und einem WebSphere-Repository oder einem anderen Repository zu speichern. 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 WebSphere 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, mit denen die Aktion des MQCONNX-Aufrufs gesteuert wird.

Hierbei handelt es sich um einen Verweis auf eine MQCNO-Struktur für Verbindungsoptionen, mit denen die Aktion des API-Aufrufs MQCONN gesteuert wird. Der Parameter ist NULL, wenn der Exit mit MQXR_TERM aufgerufen wird. Der MQI-Client stellt immer eine MQCNO-Struktur für den Exit bereit, auch wenn sie von der Anwendung ursprünglich nicht bereitgestellt wurde. Stellt eine Anwendung eine MQCNO-Struktur bereit, erstellt der Client ein Duplikat davon, das an den Exit übergeben und dort geändert wird. Der Client bleibt Eigner der MQCNO-Struktur.

Eine MQCD-Struktur, auf die in der MQCNO-Struktur verwiesen wird, hat Vorrang vor allen Verbindungsdefinitionen, die über das Array bereitgestellt werden. Der Client stellt mithilfe der MQCNO-Struktur eine Verbindung zum Warteschlangenmanager her, während die anderen Strukturen ignoriert werden.

pCompCode
Typ: PMQLONG Ein-/Ausgabe

Beendigungscode.

Verweis auf eine MQLONG-Struktur, die den Beendigungscode des Exits empfängt. Folgende Werte sind zulässig:

- `MQCC_OK` - Erfolgreiche Ausführung
- `MQCC_WARNING` - Warnung (teilweise Ausführung)
- `MQCC_FAILED` - Aufruf fehlgeschlagen

pReason
Typ: PMQLONG Ein-/Ausgabe

Ursachencode zur näheren Bestimmung von 'pCompCode'.

Verweis auf eine MQLONG-Struktur, die den Ursachencode des Exits empfängt. Lautet der Beendigungscode MQCC_OK, ist nur der folgende Wert gültig:

- MQRC_NONE - (0, x'000') Keine Ursache zurückzumelden.

Wenn der Beendigungscode MQCC_FAILED oder MQCC_WARNING lautet, kann das Feld für den Ursachencode von der Exitfunktion auf einen beliebigen MQRC_*-Wert gesetzt werden.

C Invocation

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

Zeilengruppe für PreConnect der Clientkonfigurationsdatei

Verwenden Sie die Zeilengruppe 'PreConnect', um den PreConnect-Exit in der Datei `mqclient.ini` zu konfigurieren.

Die folgenden Attribute können in die Zeilengruppe PreConnect aufgenommen werden:

`Data=`*< URL>*
URL des Repositorys, in dem Verbindungsdefinitionen gespeichert werden. Ein Beispiel bei Verwendung eines LDAP-Servers:

Data = ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com

`Function=`*<myFunc>*
Der Name des funktionalen Eingangspunkts in der Bibliothek, die den PreConnect-Exit-Code enthält.

Die Funktionsdefinition entspricht dem PreConnect-Exit-Prototyp MQ_PRECONNECT_EXIT .

Die maximale Länge dieses Feldes ist MQ_EXIT_NAME_LENGTH.

Module=*< amqldapi>*
Der Name des Moduls, das den API-Exit-Code enthält.

Wenn dieses Feld den vollständigen Pfadnamen des Moduls enthält, wird es wie angegeben verwendet.

Sequence=*< Sequenznummer>*
Die Sequenz, in der dieser Exit relativ zu anderen Exits aufgerufen wird. Ein Exit mit einer niedrigen Folgenummer wird vor einem Exit mit einer höheren Folgenummer aufgerufen. Es ist nicht erforderlich, dass die Folgenummerierung von Ausgängen stetig ist. Eine Folge von 1, 2, 3 hat das gleiche Ergebnis wie eine Folge von 7, 42, 1096. Dieses Attribut ist ein numerischer Wert ohne Vorzeichen.

Es können mehrere PreConnect-Zeilengruppen in der Datei `mqclient.ini` definiert werden. Die Verarbeitungsreihenfolge der einzelnen Exit wird durch das Attribut "Sequence" der Zeilengruppe festgelegt.

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.

Veröffentlichungsexits werden unter z/OS nicht unterstützt.

Mit dem Veröffentlichungsexit können Sie Nachrichten überprüfen und ändern, die an Subskribenten übergeben wurden:

- Inhalte einer Nachricht überprüfen, die für jeden Subskribenten veröffentlicht wird
- Inhalte einer Nachricht ändern, die für jeden Subskribenten veröffentlicht wird
- Warteschlange ändern, in die eine Nachricht eingereiht wird
- Übermittlung einer Nachricht an einen Subskribenten stoppen

Veröffentlichungsexit schreiben

Die Schritte im Abschnitt „Exits und installierbare Services schreiben und kompilieren” auf Seite 399 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.

WebSphere MQ stellt keine Implementierung des Eingangspunkts 'MQ_PUBLISH_EXIT' zur Verfügung. 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.

Möglicherweise kann eine WebSphere MQ-Verbindung, für die der Veröffentlichungsexit benötigt wird, den Exit nicht laden oder initialisieren. 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 WebSphere MQ-Ursachencode MQRC_PUBLISH_EXIT_ERROR fehl.

Der Kontext, in dem der Veröffentlichungsexit aufgerufen wird, ist die Herstellung einer Verbindung zum Warteschlangenmanager durch eine Anwendung. Der Warteschlangenmanager verwaltet einen Benutzerdatenbereich für jede Verbindung, über die Veröffentlichungsoperationen ausgeführt werden. Der Exit kann Informationen im Benutzerdatenbereich für jede Verbindung speichern.

Ein Veröffentlichungsexit kann einige MQI-Aufrufe verwenden. Es können nur die MQI-Aufrufe verwendet werden, durch die Nachrichteneigenschaften bearbeitet werden. Dies sind die folgenden Aufrufe:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Wenn der Veröffentlichungsexit den Zielwarteschlangenmanager oder den Warteschlangennamen ändert, wird keine neue Berechtigungsprüfung ausgeführt.

Veröffentlichungsexit kompilieren

Der Veröffentlichungsexit ist eine dynamisch geladene Bibliothek, die man sich als Kanalexit vorstellen kann. Weitere Informationen zum Kompilieren von Exits finden Sie unter „Exits und installierbare Services schreiben und kompilieren” auf Seite 399.

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 den WebSphere MQ -Explorer zum Definieren der Attribute verwenden. Die Attribute werden auf der Eigenschaftenseite des Warteschlangenmanagers unter 'Publish/Subscribe' definiert.

Um den Veröffentlichungsexit in der Datei qm.ini auf UNIX- und Linux-Systemen zu konfigurieren, erstellen Sie eine Zeilengruppe mit der Bezeichnung PublishSubscribe. Die Zeilengruppe PublishSubscribe enthält die folgenden Attribute:

PublishExitPath=[path]|*module_name*
Modulname und Pfad, die den Veröffentlichungsexit-Code enthalten. Die maximale Länge dieses Felds beträgt MQ_EXIT_NAME_LENGTH. Standardmäßig wird kein Veröffentlichungsexit angegeben.

PublishExitFunction=*function_name*
Name des Funktionseingangspunktes in dem Modul, das den Veröffentlichungsexit-Code enthält. Die maximale Länge dieses Felds beträgt MQ_EXIT_NAME_LENGTH.

PublishExitData=*string*
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. Um jedoch ein eigenes Benutzerexitprogramm bereitstellen zu können, mit dem das Workload-Management angepasst werden kann, enthält WebSphere MQ einen speziellen Benutzerexit, den Exit für die Clusterauslastung.

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 während der Übertragung der Nachricht fehlschlägt und die Nachricht zurückgesetzt wird, wird ein neuer Zielwarteschlangenmanager ausgewählt.

Auf allen Plattformen mit Ausnahme von z/OS lädt der Warteschlangenmanager den neuen Exit für die 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 Nicht-z/OS-Plattformen 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

Auf anderen Plattformen als z/OS dürfen Exits für die Clusterauslastung keine MQI-Aufrufe verwenden. Ansonsten entsprechen die Regeln zum Schreiben und Kompilieren von Programmen für den Exit für Clusterauslastung den Regeln, die für Kanalexitprogramme angewendet werden. Folgen Sie den Schritten im Abschnitt „Exits und installierbare Services schreiben und kompilieren” auf Seite 399 und verwenden Sie das Beispielprogramm „Beispielprogramm für Exit für Clusterauslastung” auf Seite 455, das Sie beim Schreiben und Kompilieren Ihres Exits unterstützt.

Weitere Informationen zu Kanalexits finden Sie im Abschnitt „Kanalexitprogramme schreiben” auf Seite 425.

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)
```

Beispielprogramm für Exit für Clusterauslastung

In WebSphere MQ steht Ihnen ein Beispielexitprogramm für die Clusterauslastung zur Verfügung. Sie können das Beispiel kopieren und es als Basis für Ihre eigenen Programme verwenden.

Auf anderen Plattformen als z/OS

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 57. Speicherort des Beispielexitprogramms für Clusterauslastung (gilt nicht für z/OS)

Plattform	**Dateipfad**
AIX, HP-UX, Sun Solaris	*MQ_INSTALLATION_PATH*`/samp`
Windows	*MQ_INSTALLATION_PATH*`\Tools\c\Samples`

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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:

Auf UNIX and Linux-Systemen:

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

Unter Windows:

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

Anstatt den bereitgestellten Algorithmus für das Workload-Management zu verwenden, ruft WebSphere MQ diesen Exit auf, um alle Nachrichten an den ausgewählten Warteschlangenmanager weiterzuleiten.

IBM WebSphere MQ-Anwendung erstellen

Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

Eigene Anwendung unter AIX erstellen

In den AIX-Veröffentlichungen wird 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 ausführen müssen, wenn Sie WebSphere MQ for AIX-Anwendungen zur Ausführung unter AIX erstellen. Unterstützt werden C, C++ und COBOL. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter Verwendung von C++.

Die Tasks, die Sie zum Erstellen einer ausführbaren Anwendung mithilfe von WebSphere 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 WebSphere MQ for AIX-Include-Dateien für die von Ihnen verwendete Sprache einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

Legen Sie bei Ausführung eines Thread-Servers oder einer Thread-Clientanwendung die Umgebungsvariable THREAD_SCOPE=S fest.

C-Programme unter 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 sind 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 in Codierungsstandards auf 64-Bit-Plattformen.

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

Für 32-Bit-Anwendungen:

```
$ xlc_r -o amqsput_32 amqsput0.c -IMQ_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 -IMQ_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 WebSphere MQ-Symbole aufgelöst werden.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der WebSphere MQ-MQI-Client für AIX installiert ist, müssen Sie die Programme erneut kompilieren, sodass stattdessen eine Verknüpfung mit der Clientbibliothek (`-lmqic`) hergestellt wird.

Bibliotheken verknüpfen

Sie benötigen die folgenden Bibliotheken:

- Verknüpfen Sie Ihre Programme mit der entsprechenden Bibliothek, die von WebSphere 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 WebSphere MQ-Anwendung mit Thread aus einer einzigen Kompiliereinheit führen Sie beispielsweise die folgenden Befehle aus.

Für 32-Bit-Anwendungen:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm_r
```

Dabei ist `amqsput0` ein Beispielprogramm.

Wenn Sie die Programme auf einem System verwenden möchten, auf dem nur der WebSphere MQ-MQI-Client für AIX installiert ist, müssen Sie die Programme erneut kompilieren, sodass stattdessen eine Verknüpfung mit der Clientbibliothek (`-lmqic`) hergestellt wird.

Anmerkung:

1. Wenn Sie einen installierbaren Service schreiben (weitere Informationen finden Sie unter Verwaltung), 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.
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.a` (oder `libmqmxa64.a`, falls Ihr Transaktionsmanager den Typ 'long' als 64-Bit behandelt) und `libmqz.a` in einer Anwendung ohne Thread und zu den Bibliotheken `libmqmxa_r.a` (oder `libmqmxa64_r.a`) und `libmqz_r.a` in einer Anwendung mit Thread herstellen.
3. Sie müssen vertrauenswürdige Anwendungen mit den WebSphere MQ-Threadbibliotheken verknüpfen. In WebSphere MQ auf Systemen unter UNIX and Linux kann allerdings jeweils nur eine Verbindung zu einem Thread in einer vertrauenswürdigen Anwendung hergestellt werden.
4. Sie müssen WebSphere 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 WebSphere 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 for AIX vorbereiten

COBOL-Beispielprogramme werden mit IBM WebSphere 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 \
-I<COBCPY>
```

32-Bit-Clientanwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-I<COBCPY>
```

32-Bit-Serveranwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

32-Bit-Clientanwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

64-Bit-Serveranwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -I<COBCPY>
```

64-Bit-Clientanwendung ohne Thread

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -I<COBCPY>
```

64-Bit-Serveranwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

64-Bit-Clientanwendung mit Thread

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

COBOL-Programme mithilfe von Micro Focus COBOL vorbereiten

Vor dem Kompilieren Ihres Programms legen Sie Umgebungsvariablen folgendermaßen fest:

```
export COBCPY=<COBCPY>
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 unter AIX vorbereiten

Verwenden Sie diese Informationen bei der Vorbereitung von Programmen von CICS in AIX.

XA-Switchmodule werden bereitgestellt, damit Sie CICS mit IBM WebSphere MQverknüpfen können:

Tabelle 58. 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 WebSphere MQ -Switchloaddatei *amqzsc* , die mit dem Produkt bereitgestellt wird.

Verbinden Sie Ihre C-Transaktionen immer mit der threadsicheren IBM WebSphere MQ -Bibliothek *libmqm_r.a.*, und Ihre COBOL-Transaktionen mit der COBOL-Bibliothek *libmqmcb_r.a.*.

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie unter Verwaltung .

TXSeries CICS-Unterstützung

IBM WebSphere MQ unter AIX unterstützt TXSeries CICS über die XA-Schnittstelle. Stellen Sie sicher, dass CICS-Anwendungen mit der Version mit Thread der IBM WebSphere MQ-Bibliotheken verknüpft sind.

Sie können CICS -Programme mit 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 WebSphere MQ-Programme, die in dieselbe CICS-Region geladen werden, entweder in der Programmiersprache C oder in COBOL. Es ist nicht möglich, in derselben CICS-Region eine Kombination aus MQI-Aufrufen in C und COBOL zu erstellen. Die meisten MQI-Aufrufe, die in der zweiten Sprache verwendet wurden, schlagen mit dem Ursachencode `MQRC_HOBJ_ERROR` fehl.

Vorbereiten von CICS -COBOL-Programmen mit IBM COBOL Set for AIX

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere 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  \
                   -IMQ_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 WebSphere MQ installiert ist.

Gehen Sie zur Verwendung von Micro Focus COBOL folgendermaßen vor:

1. Fügen Sie das Modul für die IBM WebSphere MQ COBOL-Laufzeitbibliothek mit folgendem Befehl zur Laufzeitbibliothek hinzu:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Anmerkung: Bei Verwendung von `cicsmkcobol` ist es in IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ installiert ist.

Erstellen Sie CICS C-Programme mit den 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-Beispieltransaktion in C

Ein Beispiel für einen C-Quellcode für eine AIX IBM WebSphere MQ-Transaktion wird in AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE auf dem Standardwarteschlangenmanager und stellt sie in die lokale Warteschlange, deren Name im Übertragungsheader der Nachricht angegeben ist. Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet. Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

Anwendung unter HP Integrity NonStop Server erstellen

In diesem Abschnitt werden die zusätzlichen Aufgaben sowie die Änderungen an den Standardaufgaben beschrieben, die Sie durchführen müssen, wenn Sie IBM WebSphere MQ-Client für HP Integrity NonStop Server-Anwendungen erstellen, die unter HP Integrity NonStop Server ausgeführt werden sollen.

C, COBOL und pTAL werden unterstützt.

OSS- und Guardian-Header und öffentliche Bibliotheken

Hier finden Sie Listen der OSS- und Guardian-Header sowie öffentlichen Bibliotheken. Aufgeführt sind OSS-Header, öffentliche ausführbare OSS-Bibliotheken und öffentliche OSS-Importbibliotheken, Guardian-Header sowie öffentliche ausführbare Guardian-Bibliotheken und öffentliche Guardian-Importbibliotheken.

„OSS-Header” auf Seite 463

„Öffentliche ausführbare OSS-Bibliotheken und öffentliche OSS-Importbibliotheken” auf Seite 463

„Guardian-Header” auf Seite 464

„Öffentliche ausführbare Guardian-Bibliotheken und öffentliche Guardian-Importbibliotheken” auf Seite 465

OSS-Header

Tabelle 59. OSS-Header

Objekt	Position	Beschreibung
cmqbc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqfc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqec.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqpsc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqxc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqzc.h	<mqinstall>/inc	IBM WebSphere MQ-C-Header (OSS)
cmqcobol.cpy	<mqinstall>/inc	IBM WebSphere MQ-COBOL-Copybook (OSS)
cmqbt.tal	<mqinstall>/inc	IBM WebSphere MQ-pTAL-Header (OSS)
cmqcft.tal	<mqinstall>/inc	IBM WebSphere MQ-pTAL-Header (OSS)
cmqpst.tal	<mqinstall>/inc	IBM WebSphere MQ-pTAL-Header (OSS)
cmqt.tal	<mqinstall>/inc	IBM WebSphere MQ-pTAL-Header (OSS)
cmqxt.tal	<mqinstall>/inc	IBM WebSphere MQ-pTAL-Header (OSS)

Öffentliche ausführbare OSS-Bibliotheken und öffentliche OSS-Importbibliotheken

Tabelle 60. Öffentliche ausführbare OSS-Bibliotheken und öffentliche OSS-Importbibliotheken

Objekt	Position	Beschreibung
libmqic.so	<mqinstall>/bin	Öffentliche ausführbare IBM WebSphere MQ-Bibliothek (OSS, ohne Threads)
libmqic_r.so	<mqinstall>/bin	Öffentliche ausführbare IBM WebSphere MQ-Bibliothek (OSS, Multithread)
libmqic.so	<mqinstall>/lib	Öffentliche IBM WebSphere MQ-Importbibliothek (OSS, ohne Threads)

Tabelle 60. Öffentliche ausführbare OSS-Bibliotheken und öffentliche OSS-Importbibliotheken (Forts.)		
Objekt	**Position**	**Beschreibung**
libmqic_r.so	<mqinstall>/lib	Öffentliche IBM WebSphere MQ-Importbibliothek (OSS, Multithread)
mqicb	<mqinstall>/lib	Öffentliche IBM WebSphere MQ-Importbibliothek für COBOL (OSS)

Guardian-Header

Tabelle 61. Guardian-Header		
Objekt	**Position**	**Beschreibung**
cmqbch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqfch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqech	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqpsch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqxch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqzch	<mqinstall>/inc/G	IBM WebSphere MQ-C-Header (Guardian)
cmqcobol	<mqinstall>/inc/G	IBM WebSphere MQ-COBOL-Copybook (Guardian)
cmqbt	<mqinstall>/inc/G	IBM WebSphere MQ-pTAL-Header (Guardian)
cmqcft	<mqinstall>/inc/G	IBM WebSphere MQ-pTAL-Header (Guardian)
cmqpst	<mqinstall>/inc/G	IBM WebSphere MQ-pTAL-Header (Guardian)
cmqt	<mqinstall>/inc/G	IBM WebSphere MQ-pTAL-Header (Guardian)
cmqxt	<mqinstall>/inc/G	IBM WebSphere MQ-pTAL-Header (Guardian)

Öffentliche ausführbare Guardian-Bibliotheken und öffentliche Guardian-Import-bibliotheken

Tabelle 62. Öffentliche ausführbare Guardian-Bibliotheken und öffentliche Guardian-Importbibliotheken		
Objekt	**Position**	**Beschreibung**
mqic	<mqinstall>/bin/G	Öffentliche ausführbare IBM WebSphere MQ-Bibliothek (Guardian)
mqicb	<mqinstall>/lib/G	Öffentliche IBM WebSphere MQ-Importbibliothek für COBOL (Guardian)

C-Programme in HP Integrity NonStop Server vorbereiten

Dieser Abschnitt enthält Informationen zur Vorbereitung von C-Programmen in HP Integrity NonStop Server sowie Beispiele der bei Verwendung des OSS-C-Compilers und des Guardian-C-Compilers beim Erstellen von Anwendungen zu verwendenden Befehle.

Im Verzeichnis MQ_INSTALLATION_PATH/opt/mqm/samp/bin sind vorkompilierte C-Programme zu finden. Verwenden Sie den c89-Compiler, um ein Beispiel aus Quellcode zu erstellen.

Sie müssen Ihre Programme müssen mit der entsprechenden von IBM WebSphere MQ bereitgestellten Bibliothek verknüpfen. In der folgenden Tabelle sind die Bibliotheken aufgeführt, zu denen eine Verknüpfung hergestellt werden muss, wenn C-Programme in HP Integrity NonStop Server vorbereitet werden.

Tabelle 63. . HP Integrity NonStop Server-Verbindungsbibliotheken	
Bibliothek	**Beschreibung**
libmqic.so	OSS ohne Threads
libmqic_r.so	OSS als Multithread
mqic	Wächter

Native IBM WebSphere MQ-Multithreadanwendungen müssen die Funktion 'Posix User Threads (PUT)' verwenden. 'Standard Posix Threads (SPT)' wird in diesem Produkt nicht unterstützt.

Anwendungen mit dem OSS-C-Compiler erstellen

Dieser Abschnitt enthält Beispiele der Befehle, die bei Verwendung des OSS-Compilers zum Erstellen von Programmen verwendet werden, die für OSS oder Guardian bestimmt sind.

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

Im folgenden Beispiel wird eine C-Client-OSS-Anwendung ohne Threads erstellt:

```
c89 -Wsystype=oss -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
```

Im folgenden Beispiel wird eine Multithread-C-Client-OSS-Anwendung erstellt:

```
c89 -Wsystype=oss -D_PUT_MODEL_ -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic_r -lput
```

Im folgenden Beispiel wird eine Guardian-C-Client-Anwendung erstellt:

```
c89 -Wsystype=guardian -o /G/vol/subvol/amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
```

Anwendungen mit dem Guardian-C-Compiler erstellen

Dieser Abschnitt enthält Beispiele der Befehle, die bei Verwendung des Guardian-Compilers zum Erstellen von Programmen verwendet werden, die für Guardian bestimmt sind.

MQ_INSTALLATION_PATH stellt den Guardian-Datenträger und -Unterdatenträger dar, in dem IBM WebSphere MQ installiert ist.

Im folgenden Beispiel wird eine Guardian-C-Client-Anwendung erstellt:

```
CCOMP /in AMQSPUT0/ AMQSPUTC;&
       runnable,systype guardian,nolist,&
       ssv0 "$system.system",&
       ssv1 "MQINSTALLATION_SUBVOL",&
       ld(-LMQINSTALLATION_SUBVOL -lmqic)
```

COBOL-Programme vorbereiten

In diesem Abschnitt finden Sie Informationen zur Vorbereitung von C-Programmen auf den IBM WebSphere MQ-Client für HP Integrity NonStop Server. Der Abschnitt enthält Beispiele der beim Erstellen von Anwendungen mit dem OSS-ECOBOL-Compiler und dem Guardian-ECOBOL-Compiler zu verwendenden Befehle.

Verwenden Sie den ECOBOL-Compiler, um ein COBOL-Beispiel aus Quellcode zu erstellen.

In der folgenden Tabelle sind die Bibliotheken aufgeführt, die benötigt werden, wenn COBOL-Programme in HP Integrity NonStop Server vorbereitet werden. Sie müssen Ihre Programme müssen mit der entsprechenden von IBM WebSphere MQ bereitgestellten Bibliothek verknüpfen.

Tabelle 64. . HP Integrity NonStop Server-Verbindungsbibliotheken

Bibliothek	**Beschreibung**
libmqic.so	OSS ohne Threads
mqic	Wächter

Wenn Sie eine COBOL-Anwendung ausführen, die eine Verbindung zu einem Warteschlangenmanager herstellt, müssen Sie zuerst die Variable *SAVE-ENVIRONMENT* auf ON setzen. So setzen Sie die Variable *SAVE-ENVIRONMENT* auf ON:

- Geben Sie für OSS folgenden Befehl ein:

```
export SAVE-ENVIRONMENT=ON
```

- Geben Sie für Guardian folgenden Befehl ein:

```
param SAVE-ENVIRONMENT ON
```

Wenn Sie die Variable *SAVE-ENVIRONMENT* nicht auf ON setzen, schlägt die Anwendung bei dem Versuch, eine Verbindung zu einem Warteschlangenmanager herzustellen, mit dem Ursachencode 2058 (080A) (RC2058): MQRC_Q_MGR_NAME_ERROR fehl.

Anwendungen mit dem OSS-ECOBOL-Compiler erstellen

Dieser Abschnitt enthält Beispiele der Befehle, die bei Verwendung des OSS-ECOBOL-Compilers zum Erstellen von Programmen verwendet werden, die für OSS oder Guardian bestimmt sind.

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

Im folgenden Beispiel wird eine COBOL-Client-OSS-Anwendung erstellt:

```
   ecobol -Wsystype=oss
          -Wcobol="ansi;port"
          -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
```

```
        -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
        -LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
        -o amq0put0
        MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

Im folgenden Beispiel wird eine COBOL-Client-Guardian-Anwendung erstellt:

```
ecobol -Wsystype=guardian
       -Wcobol="ansi;port;save all"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

Anwendungen mit dem Guardian-ECOBOL-Compiler erstellen

Dieser Abschnitt enthält Beispiele der Befehle, die bei Verwendung des Guardian-ECOBOL-Compilers zum Erstellen von Programmen verwendet werden, die für Guardian bestimmt sind.

MQ_INSTALLATION_SUBVOL stellt den Guardian-Datenträger und -Unterdatenträger dar, in dem IBM WebSphere MQ installiert ist.

Im folgenden Beispiel wird eine COBOL-Client-Guardian-Anwendung erstellt:

```
ECOBOL /in MQSPUTL/ MQSPUT,MQINSTALLATION_SUBVOL.cmqcobol;
       call-shared;ansi;port;save all;nolist;runnable;
       consult MQINSTALLATION_SUBVOL.mqicb;
       eld(-LMQINSTALLATION_SUBVOL -lmqic)
```

pTAL-Programme vorbereiten

Hier erfahren Sie, wie Sie pTAL-Programme für den IBM WebSphere MQ-Client auf der HP Integrity NonStop Server-Plattform erstellen können.

Verwenden Sie den EPTAL-Compiler, um ein pTAL-Beispiel aus Quellcode zu erstellen.

Anmerkung:

- IBM WebSphere MQ-pTAL-Anwendungen müssen eine in C oder COBOL geschriebene Hauptroutine verwenden.
- Das Erstellen von pTAL-Anwendungen ist ausschließlich in Guardian möglich.

In der folgenden Tabelle ist die bei der Vorbereitung von pTAL-Programmen in HP Integrity NonStop Server erforderliche Bibliothek aufgeführt. Sie müssen Ihre Programme müssen mit der entsprechenden von IBM WebSphere MQ bereitgestellten Bibliothek verknüpfen.

Tabelle 65. . HP Integrity NonStop Server-Verbindungsbibliothek

Bibliothek	**Beschreibung**
mqic	Wächter

Anwendungen mit dem Guardian-EPTAL-Compiler erstellen

Dieser Abschnitt enthält Beispiele der Befehle, die bei Verwendung des Guardian-EPTAL-Compilers zum Erstellen von Programmen verwendet werden, die für Guardian bestimmt sind.

MQINSTALLATION_SUBVOL stellt den Guardian-Datenträger und -Unterdatenträger dar, in dem IBM WebSphere MQ installiert ist.

IBM WebSphere MQ-pTAL-Anwendungen müssen eine in C oder COBOL geschriebene Hauptroutine verwenden.

Im folgenden Beispiel wird eine pTAL-Client-Guardian-Anwendung erstellt:

```
ASSIGN SSV0, $SYSTEM.SYSTEM
ASSIGN SSV1, MQINSTALLATION_SUBVOL

EPTAL /in MQINSTALLATION_SUBVOL.MQSPUTT/ MQSPUTO;nolist

CCOMP /in MQINSTALLATION_SUBVOL.MQSPTMC/ MQSPUT;
       runnable,systype guardian,extensions,nolist,
       ssv0 "$system.system",
       ssv1 "MQINSTALLATION_SUBVOL",
       eld(MQSPUTO -LMQINSTALLATION_SUBVOL -lmqic)
```

Anwendung unter HP-UX erstellen

In diesem Abschnitt werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie ausführen müssen, wenn Sie WebSphere MQ for HP-UX-Anwendungen zur Ausführung unter HP-UX erstellen.

Unterstützt werden C, C++ und COBOL. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter Verwendung von C++.

Die Tasks, die Sie zum Erstellen einer ausführbaren Anwendung mithilfe von WebSphere 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 WebSphere MQ for HP-UX-Include-Dateien für die von Ihnen verwendete Sprache einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

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 unter 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 WebSphere 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.

Zur Verwendung von SSL müssen WebSphere MQ-MQI-Clients unter HP-UX mit POSIX-Threads erstellt werden.

Folgende Beispiele sollten beachtet werden:

- „IA64-Plattform (IPF)” auf Seite 468
- „Bibliotheken verknüpfen” auf Seite 470

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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
  -LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld  -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -LMQ_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 -IMQ_INSTALLATION_PATHMQ_INSTALLATI
ON_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
ld  -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Bibliotheken verknüpfen

Sie müssen Ihre Programme mit der entsprechenden Bibliothek verknüpfen, die von WebSphere MQ bereitgestellt wird.

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. Wenn Sie einen installierbaren Service schreiben (weitere Informationen dazu finden Sie unter Verwaltung), müssen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.sl` herstellen.
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.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.
3. Sie müssen WebSphere MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

COBOL-Programme unter HP-UX vorbereiten

Hier finden Sie Informationen zur Vorbereitung von COBOL-Programmen unter HP-UX mithilfe von Micro Focus Server Express mit WebSphere MQ auf der IA64-Plattform (IPF) und zur Ausführung von Programmen in der WebSphere MQ-MQI-Clientumgebung.

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

Hinweise für Benutzer

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>"
```

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 WebSphere 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 WebSphere MQ auf der IA64-Plattform (IPF) verwenden

Unter „Von WebSphere MQ for HP-UX auf IA64 (IPF) unterstützte Adressraummodelle” auf Seite 473 finden Sie Einzelheiten zur Verwendung von Micro Focus Server Express in Verbindung mit WebSphere MQ auf der HP/IPF-Plattform.

Programme zur Ausführung in WebSphere MQ-MQI-Clientumgebung

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 unter HP-UX vorbereiten

Erfahren Sie, wie CICS-Transaktionsprogramme unter HP-UX erstellt werden.

Zum Erstellen der CICS-Beispieltransaktion 'amqscic0.ccs' führen Sie den folgenden Befehl aus:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

Über ein bereitgestelltes XA-Switch-Modul können Sie CICS mit WebSphere MQ verknüpfen:

Tabelle 66. 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 in Verwaltung .

TXSeries CICS-Unterstützung

WebSphere MQ on HP-UX unterstützt TXSeries CICS durch die Verwendung der XA-Schnittstelle. Vergewissern Sie sich, dass die CICS-Anwendungen mit der Version mit Thread der MQ-Bibliotheken verknüpft sind.

Schreiben Sie WebSphere MQ-Programme, die in dieselbe CICS-Region geladen werden, entweder in der Programmiersprache C oder in COBOL. Es ist nicht möglich, in derselben CICS-Region eine Kombination aus MQI-Aufrufen in C und COBOL zu erstellen. Die meisten MQI-Aufrufe, die in der zweiten Sprache verwendet wurden, schlagen mit dem Ursachencode `MQRC_HOBJ_ERROR` fehl.

CICS-Beispieltransaktion in C

Die C-Beispielquelle für eine CICS WebSphere MQ-Transaktion wird von AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORK-QUEUE auf dem Standardwarteschlangenmanager und stellt sie in die lokale Warteschlange, deren Name

im Übertragungsheader der Nachricht angegeben ist. 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 WebSphere MQ installiert ist.

Gehen Sie zur Verwendung von Micro Focus COBOL folgendermaßen vor:

1. Fügen Sie das WebSphere MQ-COBOL-Laufzeitbibliotheksmodul mit dem folgenden Befehl zur Laufzeitbibliothek hinzu:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Anmerkung: Bei Verwendung von `cicsmkcobol` ist es in WebSphere MQ nicht möglich, von der COBOL-Anwendung aus MQI-Aufrufe in der Programmiersprache C auszugeben.

   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. Nachdem Sie diese Funktionen verschoben haben, schließen Sie die WebSphere MQ-Bibliothek `libmqmcbrt.o` nicht ein, wenn Sie die COBOL-Anwendung für CICS erstellen.

   Wenn Ihre COBOL-Anwendung keine COBOL MQI-Aufrufe vornimmt, stellen Sie außerdem keine Verknüpfung zwischen `libmqmz_r` und `cicsmkcobol` her.

   Damit wird die Micro Focus COBOL-Sprachmethodendatei erstellt und es wird der CICS-Laufzeit-COBOL-Bibliothek ermöglicht, WebSphere MQ auf Systemen unter UNIX and Linux aufzurufen.

   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 WebSphere 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 WebSphere MQ for HP-UX auf IA64 (IPF) unterstützte Adressraummodelle

Unter HP-UX werden mehrere Modelle für Adressräume bereitgestellt, die von WebSphere MQ-Anwendungen genutzt werden können.

HP-UX unterstützt zwei Adressraummodelle:

- MGAS - Mostly Global Address Space (dies ist die Standardeinstellung, die von WebSphere MQ verwendet wird)
- MPAS - Mostly Private Address Space

Anwendungen, die eine Verbindung zu WebSphere MQ herstellen, können entweder das Adressraummodell MGAS oder MPAS verwenden. Bei Anwendungen, die anhand des MPAS-Modells erstellt wurden und

eine Verbindung zu WebSphere MQ mithilfe eines gemeinsam genutzten Speichers herstellen, kann es aufgrund der Ineffizienz bei der Zuordnung der von WebSphere MQ verwendeten Seiten im 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 WebSphere 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 Einzelheiten zu den Adressraummodellen MGAS und MPAS sind in der HP-UX-Dokumentation zu finden.

Anwendung unter Linux erstellen

In diesen Informationen werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie ausführen müssen, wenn Sie Anwendungen von WebSphere MQ for Linux für die Ausführung erstellen.

C und C++ werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter Verwendung von C++.

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 WebSphere 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 WebSphere 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.so	Server für C
libmqic.so & libmqm.so	Client für C

  Stellen Sie in einer Threadumgebung eine Verknüpfung zu einer 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. Wenn Sie einen installierbaren Service schreiben (weitere Informationen dazu finden Sie unter Verwaltung), müssen Sie eine Verknüpfung zu der Bibliothek `libmqmzf.so` herstellen.
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` in einer Anwendung ohne Thread bzw. zu den Bibliotheken `libmqmxa_r.so` (oder `libmqmxa64_r.so`) und `libmqz_r.so` in einer Anwendung mit Thread herstellen.
3. Sie müssen WebSphere 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 WebSphere MQ installiert ist.

C-Clientanwendung, 31-Bit, kein Thread

```
gcc -m31  -o famqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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 WebSphere MQ installiert ist.

C-Clientanwendung, 32-Bit, kein Thread

```
gcc -m32 -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
IMQ_INSTALLATION_PATH/inc -LMQ_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 WebSphere MQ installiert ist.

C-Clientanwendung, 64-Bit, kein Thread

```
gcc -m64  -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

COBOL-Programme in Linux vorbereiten

Hier finden Sie Informationen zum Vorbereiten von COBOL-Programmen in Linux und zum Vorbereiten von COBOL-Programmen mithilfe von Micro Focus COBOL.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere 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. Auf 64-Bit-Plattformen sind COBOL-Copybooks im 64-Bit-Format 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.

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 mithilfe von Micro Focus COBOL vorbereiten

Vor dem Kompilieren Ihres Programms legen Sie Umgebungsvariablen folgendermaßen fest:

```
export COBCPY=<COBCPY>
export LIB=MQ_INSTALLATION_PATHlib:$LIB
```

Zum Kompilieren eines 32-Bit-COBOL-Programms (falls unterstützt) mithilfe von Micro Focus COBOL geben Sie Folgendes ein:

```
$ 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
```

Zum Kompilieren eines 64-Bit-COBOL-Programms mithilfe von Micro Focus COBOL geben Sie Folgendes ein:

```
$ 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.

Anwendung unter Solaris erstellen

In diesem Abschnitt werden die zusätzlichen Tasks und die Änderungen an den Standardtasks beschrieben, die Sie ausführen müssen, wenn Sie WebSphere MQ for Solaris-Anwendungen zur Ausführung unter Solaris erstellen.

Die Programmiersprachen COBOL, C und C++ werden unterstützt. Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter Verwendung von C++.

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 WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

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 unter Solaris vorbereiten

Vorkompilierte C-Programme werden im Verzeichnis *MQ_INSTALLATION_PATH*`/samp/bin` bereitgestellt.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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 WebSphere MQ MQI-Client für Solaris installiert ist, müssen Sie die Programme kompilieren, sodass eine Verknüpfung mit der Clientbibliothek (`-lmqic`) hergestellt wird.

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 mit FIPS 140-2-konformen Betrieb konfigurierten 32 Bit Solaris x86-SSL- und TLS-Clients schlagen fehl, wenn sie auf Intel-Systemen ausgeführt werden. Dieser Fehler tritt auf, weil 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. Sie können dieses Problem beheben, indem Sie die FIPS 140-2-Konformität inaktivieren oder die Clientanwendung für 64 Bit kompilieren, da 64-Bit-Code nicht betroffen ist.

Bibliotheken verknüpfen

Sie müssen eine Verknüpfung zu den WebSphere 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 Verwaltung), 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 WebSphere 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 WebSphere MQ installiert ist.

C-Clientanwendung, 32-Bit

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C-Clientanwendung, 64-Bit

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

C-Serveranwendung, 32-Bit

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C-Serveranwendung, 64-Bit

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

C++-Clientanwendung, 32-Bit

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_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 WebSphere MQ installiert ist.

C-Clientanwendung, 32-Bit

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C-Clientanwendung, 64-Bit

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C-Serveranwendung, 32-Bit

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C-Serveranwendung, 64-Bit

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

C++-Clientanwendung, 32-Bit

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATI
ON_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

COBOL-Programme unter Solaris vorbereiten

Hier finden Sie Informationen zur Vorbereitung von COBOL-Programmen unter Solaris.

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere 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>"
```

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 unter Solaris vorbereiten

Hier finden Sie Informationen zur Vorbereitung von CICS-Programmen in Solaris.

Über ein bereitgestelltes XA-Switch-Modul können Sie CICS mit WebSphere MQ verknüpfen:

Tabelle 67. 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 WebSphere MQ-Bibliothek 'libmqm.so'.

Weitere Informationen zu unterstützenden CICS-Transaktionen finden Sie in Verwaltung .

Unterstützung für TXSeries CICS

WebSphere MQ for Solaris unterstützt TXSeries CICS über die XA-Schnittstelle.

Schreiben Sie WebSphere MQ-Programme, die in dieselbe CICS-Region geladen werden, entweder in der Programmiersprache C oder in COBOL. Es ist nicht möglich, in derselben CICS-Region eine Kombination aus MQI-Aufrufen in C und COBOL zu erstellen. 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 WebSphere MQ installiert ist.

Gehen Sie zur Verwendung von Micro Focus COBOL folgendermaßen vor:

1. Fügen Sie das WebSphere MQ-COBOL-Laufzeitbibliotheksmodul mit dem folgenden Befehl zur Laufzeitbibliothek hinzu:

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
```

   Anmerkung: Bei Verwendung von `cicsmkcobol` ist es in WebSphere MQ nicht möglich, von der COBOL-Anwendung aus MQI-Aufrufe in der Programmiersprache C auszugeben.

   Wenn Ihre vorhandene Anwendung solche Aufrufe enthält, verschieben Sie diese Funktionen aus den COBOL-Anwendungen in Ihre eigene Bibliothek (z. B. `myMQ.so`). Nachdem Sie diese Funktionen verschoben haben, dürfen Sie die WebSphere MQ-Bibliothek `libmqmcbrt.o` nicht einschließen, wenn Sie die COBOL-Anwendung für CICS erstellen.

   Wenn Ihre COBOL-Anwendung keine COBOL MQI-Aufrufe vornimmt, stellen Sie außerdem keine Verknüpfung zwischen `libmqmz_r` und `cicsmkcobol` her.

   Damit wird die Micro Focus COBOL-Sprachmethodendatei erstellt und es wird der CICS-Laufzeit-COBOL-Bibliothek ermöglicht, WebSphere MQ auf Systemen unter UNIX and Linux aufzurufen.

   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 WebSphere 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 unter Verwendung 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-Beispielstransaktion in C

Die C-Beispielquelle für eine CICS WebSphere MQ-Transaktion wird von AMQSCIC0.CCS bereitgestellt. Die Transaktion liest Nachrichten aus der Übertragungswarteschlange SYSTEM.SAMPLE.CICS.WORKQUEUE auf dem Standardwarteschlangenmanager und stellt sie in die lokale Warteschlange, deren Name im Übertragungsheader der Nachricht angegeben ist. Fehler werden an die Warteschlange SYSTEM.SAMPLE.CICS.DLQ gesendet. Erstellen Sie mithilfe des MQSC-Beispielscripts AMQSCIC0.TST diese Warteschlangen und die Beispieleingabewarteschlangen.

Eigene Anwendung auf Windows-Systemen erstellen

In den Veröffentlichungen für Windows-Systeme wird 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 ausführen müssen, wenn Sie WebSphere MQ for Windows-Anwendungen zur Ausführung auf Windows-Systemen erstellen. Die Programmiersprachen ActiveX, C, C++, COBOL und Visual Basic werden unterstützt. Informationen zur Vorbereitung Ihrer ActiveX-Programme finden Sie unter Verwendung der Component Object Model-Schnittstelle (WebSphere MQ Automation Classes for ActiveX). Weitere Informationen zur Vorbereitung Ihrer C++-Programme finden Sie unter Verwendung von C++.

Die Tasks, die Sie zum Erstellen einer ausführbaren Anwendung mithilfe von WebSphere 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 WebSphere MQ for Windows-Include-Dateien für die von Ihnen verwendete Sprache einzuschließen. Machen Sie sich mit den Inhalten dieser Dateien vertraut. Eine vollständige Beschreibung finden Sie unter „IBM WebSphere MQ-Datendefinitionsdateien” auf Seite 84.

64-Bit-Anwendungen unter Windows erstellen

Unter IBM WebSphere MQ for Windows Version 7.5 werden 32-Bit- und 64-Bit-Anwendung unterstützt. Die ausführbaren Dateien und Bibliotheksdateien für IBM WebSphere 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 WebSphere MQ-Bibliotheken werden in den folgenden Positionen bereitgestellt:

Tabelle 68. Position der IBM WebSphere MQ-Bibliotheken

Bibliotheksversion	**Verzeichnis mit den Bibliotheksdateien**
32 Bit	*MQ_INSTALLATION_PATH*\Tools\Lib
64 Bit	*MQ_INSTALLATION_PATH*\Tools\Lib64

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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 unter Windows vorbereiten

Arbeiten Sie in Ihrer typischen Windows-Umgebung. Für WebSphere MQ for Windows sind keine besonderen Voraussetzungen zu erfüllen.

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 WebSphere MQ bereitgestellt werden:

Bibliotheksdatei	Programm/Exit-Typ
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server für 32-Bit-C
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client für 32-Bit-C
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Client für 32-Bit-C mit Transaktionskoordination
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server für 64-Bit-C
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client für 64-Bit-C
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Client für 64-Bit-C mit Transaktionskoordination

MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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 Verwaltung), 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 WebSphere MQ-Bibliotheken vor allen anderen Produktbibliotheken verknüpfen.

- Die DLLs müssen sich in dem von Ihnen angegebenen Pfad (PATH) befinden.
- Wenn Sie Kleinbuchstaben verwenden, wann immer dies möglich ist, können Sie von WebSphere MQ for Windows-Systemen zu WebSphere MQ-Systemen unter UNIX and Linux wechseln, auf denen die Verwendung von Kleinbuchstaben erforderlich ist.

CICS-Programme und Transaktionsserverprogramme vorbereiten

Die C-Beispielquelle für eine CICS WebSphere MQ-Transaktion wird von AMQSCIC0.CCS bereitgestellt. Sie wird mithilfe der CICS-Standardfunktionen erstellt. Beispiel für TXSeries for Windows 2000:

1. Legen Sie die Umgebungsvariable fest (geben Sie den folgenden Code in einer Zeile ein):

   ```
   set CICS_IBMC_FLAGS=-IMQ_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 WebSphere 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 in Verwaltung .

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 COBOL-Copybooks im 32-Bit-Format sind im folgenden Verzeichnis installiert: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. Die COBOL-Copybooks im 64-Bit-Format 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 auf

   ```
   CopyBook64
   ```

   für 64-Bit-Anwendungen.

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

Zur Vorbereitung von COBOL-Programmen auf Windows-Systemen verknüpfen Sie Ihr Programm mit einer der folgenden Bibliotheken, die von IBM WebSphere MQ bereitgestellt werden:

Bibliotheksdatei	Programm oder Exittyp
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb	32-Bit-Server für IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb	32-Bit-Server für Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb	32-Bit-Client für IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb	32-Bit-Client für Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb	64-Bit-Server für IBM COBOL

Bibliotheksdatei	Programm oder Exittyp
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb	64-Bit-Server für Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb	64-Bit-Client für IBM 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 WebSphere MQ-Bibliothek angezeigt wird.

Abhängig vom Programm können Sie den IBM COBOL Set-Compiler oder den Micro Focus COBOL-Compiler verwenden:

- Programme, die mit `amqi` beginnen, sind für den IBM COBOL Set-Compiler geeignet,
- Programme, die mit `amqm` beginnen, sind für den Micro Focus COBOL-Compiler geeignet, und
- Programme, die mit `amq0` beginnen, sind für beide Compiler geeignet.

IBM und Micro Focus COBOL

Verwenden Sie anstelle der Bibliotheken `mqmcbb` und `mqiccbb` die Bibliothek `mqmcb.lib` oder `mqiccb.lib`, um eine erneute Verbindung zu einer 32-Bit-Version eines vorhandenen COBOL-Programms für IBM WebSphere MQ Micro Focus herzustellen.

So kompilieren Sie z. B. das Beispielprogramm `amq0put0` mithilfe von IBM VisualAge COBOL:

1. Legen Sie die Umgebungsvariable SYSLIB so fest, dass der Pfad zu den IBM WebSphere MQ VisualAge-COBOL-Copybooks eingeschlossen ist (geben Sie den folgenden Code in einer Zeile ein):

   ```
   set SYSLIB=MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook\VAcobol;%SYSLIB%
   ```

2. Für die Verwendung auf dem IBM WebSphere MQ-Server:

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqmcbb.lib"
   ```

3. Für die Verwendung auf dem IBM WebSphere MQ-Client:

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqiccbb.lib"
   ```

   Anmerkung: Obwohl Sie die Compileroption `CALLINT(SYSTEM)` verwenden müssen, ist dies die Standardeinstellung für `cob2`.

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 WebSphere 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 zur Verwendung auf dem IBM WebSphere MQ-Server:

```
cbllink amq0put0.obj mqmcb.lib
```

- Alternativ verknüpfen Sie die Objektdatei für die Verwendung auf dem IBM WebSphere 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
```

Visual Basic-Programme unter Windows vorbereiten

Lesen Sie diese Informationen, wenn Sie in Erwägung ziehen, Visual Basic-Programme unter Windows zu verwenden.

Anmerkung: Es werden keine 64-Bit-Versionen der Visual Basic-Moduldateien geliefert.

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-Support
CMQXB.BAS	Kanalexit-Support
CMQPSB.BAS	Publish/Subscribe

Unter „Codierung in Visual Basic” auf Seite 91 finden Sie Informationen zur Verwendung des MQCONNXAny-Aufrufs aus 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 WebSphere MQ-Server oder -Client erstellen, indem Sie die Variable *MqType* für die bedingte Kompilierung festlegen. Setzen Sie in einem Visual Basic-Projekt die Variable *MqType* für einen Server auf 1 und für einen Client auf 2. Gehen Sie dabei wie folgt vor:

1. Wählen Sie das Menü 'Project' (Projekt) aus.
2. Wählen Sie Eigenschaften für *Name* aus (wobei *Name* für den Namen des aktuellen Projekts steht).
3. Wählen Sie im Dialogfeld die Registerkarte 'Make' (Erstellen) aus.
4. Geben Sie im Feld 'Conditional Compilation Arguments' (Argumente für bedingte Kompilierung) für einen Server Folgendes ein:

   ```
   MqType=1
   ```

   Geben Sie in diesem Feld für einen Client Folgendes ein:

   ```
   MqType=2
   ```

SSPI-Sicherheitsexit

WebSphere MQ for Windows stellt einen Sicherheitsexit für den WebSphere MQ-MQI-Client und den WebSphere MQ-Server zur Verfügung. Dabei handelt es sich um ein Kanalexitprogramm zur Authentifizierung von WebSphere MQ-Kanälen mithilfe der SSPI-Schnittstelle (SSPI = Security Services Programming Interface). Die SSPI stellt die integrierten Sicherheitsfunktionen von Windows-Systemen bereit.

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 „Verwenden des SSPI-Sicherheitsexits auf Windows-Systemen” auf Seite 179.

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 69. 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 „WebSphere MQ-Zugriffssteuerung und Windows-Prinzipale” auf Seite 494). 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.

WebSphere MQ-Zugriffssteuerung und Windows-Prinzipale

Die von WebSphere 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 verläuft, übergibt der Exit die dem Windows-Prinzipal zugeordnete Benutzer-ID zur Zugriffssteuerung an WebSphere MQ.

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 zum Warteschlangenmanagernamen gebildet wird.

  ```
  ibmMQSeries/
  ```

LDAP-Services mit WebSphere MQ for Windows verwenden

In diesem Abschnitt wird der Verzeichnisservice erläutert und Sie erhalten Informationen darüber, welche Rolle ein Verzeichniszugriffsprotokoll (DAP = Directory Access Protocol) spielt. Darüber hinaus wird in diesem Abschnitt anhand eines Beispielprogramms erklärt, wie WebSphere MQ-Anwendungen ein LDAP-Verzeichnis (LDAP = Lightweight Directory Access Protocol) verwenden können.

Anmerkung: Das Beispielprogramm ist für Benutzer konzipiert, die bereits über LDAP-Kenntnisse verfügen.

Die folgenden Abschnitte enthalten weiterführende Informationen zu den Verzeichnisservices, zu LDAP und zur Verwendung von LDAP mit WebSphere MQ.

- „Verzeichnisservice” auf Seite 494
- „Lightweight Directory Access Protocol (LDAP)” auf Seite 495
- „LDAP mit WebSphere MQ verwenden” auf Seite 495

Verzeichnisservice

Bei einem Verzeichnis handelt es sich um ein Repository mit Informationen zu Objekten, das so organisiert ist, dass das Auffinden von Informationen zu einem bestimmten Objekt einfach ist.

Ein allgemeines Beispiel ist ein Telefonverzeichnis, in dem Informationen (Adressen und Telefonnummern) zu Personen und Unternehmen gespeichert werden. Ein weiteres Beispiel ist ein Adressbuch

für ein E-Mail-System, in dem E-Mail-Adressen und optional weitere Informationen wie beispielsweise Telefonnummern zu Personen gespeichert sind.

Auf Computersystemen können in Verzeichnissen Informationen zu Computerressourcen (z. B. zu Druckern oder gemeinsam genutzten Platten) gespeichert werden. Sie können ein Verzeichnis z. B. verwenden, um festzustellen, wo sich der nächste erreichbare Farbdrucker befindet. In einer WebSphere MQ-Anwendung kann ein Verzeichnis verwendet werden, um eine Zuordnung zwischen einem Anwendungsservice (z. B. zur Verarbeitung von Forderungen) und der Warteschlange herzustellen, die für Nachrichten verwendet wird, für die dieser Service benötigt wird (und die z. B. durch den Warteschlangennamen und den Namen des zugehörigen Host-Warteschlangenmanagers angegeben wird).

Verzeichnisse sind als Client/Server-Systeme implementiert, wobei der Verzeichnisserver alle Informationen enthält und die Anforderungen von Clients beantwortet. Bei den Clients kann es sich um Benutzerschnittstellenprogramme handeln, die Informationen direkt an den Benutzer liefern, oder um Anwendungsprogramme, die zur Ausführung der vorgesehenen Operationen bestimmte Ressourcen lokalisieren müssen. Ein Verzeichnisservice umfasst den Verzeichnisserver, die Administrationsprogramme und die Clientbibliotheken und -programme, die zum Konfigurieren, Aktualisieren und Lesen des Verzeichnisses erforderlich sind.

Lightweight Directory Access Protocol (LDAP)

Es gibt zahlreiche Verzeichnisservices wie beispielsweise Novell Directory Services, DCE Cell Directory Service, Banyan StreetTalk, Windows Directory Services, X.500 und die Adressbuchservices, die den verschiedenen E-Mail-Produkten zugeordnet sind. X.500 wurde von der International Standards Organisation (ISO) als Standard für globale Verzeichnisservices vorgeschlagen. Zur Verwendung dieses Produkts ist ein OSI-Protokollstack für die Kommunikation erforderlich. Hauptsächlich aus diesem Grund ist seine Nutzung auf große Unternehmen und akademische Einrichtungen beschränkt. Ein X.500-Verzeichnisserver kommuniziert mit seinen Clients über DAP (Directory Access Protocol).

LDAP (Lightweight Directory Access Protocol) wurde als vereinfachte Version von DAP erstellt. Dieses Protokoll ist einfacher zu implementieren, wurde um einige der weniger häufig verwendeten Funktionen von DAP bereinigt und wird über TCP/IP ausgeführt. Aufgrund dieser Änderungen hat es sich rasch zu dem Verzeichniszugriffsprotokoll für ein breites Einsatzspektrum entwickelt und bietet Ersatz für die Vielzahl proprietärer Protokolle, die in der Vergangenheit genutzt wurden. LDAP-Clients können weiterhin über ein Gateway auf einen X.500-Server zugreifen (X.500 benötigt weiterhin den OSI-Protokollstack). Die verfügbaren X.500-Implementierungen umfassen zunehmend eine native Unterstützung für den LDAP- sowie für den DAP-Zugriff.

LDAP-Verzeichnisse können verteilt werden und die Replikation verwenden, um den effizienten Zugriff auf ihre Inhalte zu ermöglichen.

Eine ausführlichere Beschreibung von LDAP finden Sie in der Veröffentlichung *Understanding LDAP*, einer IBM Redbooks .

LDAP mit WebSphere MQ verwenden

In WebSphere MQ-Konfigurationen werden die Informationen, mit denen Nachrichten- und Übertragungswarteschlangen definiert werden, lokal gespeichert. Dies bedeutet, dass in einem WebSphere MQ-Netz die verschiedenen Definitionen verteilt werden, wobei kein zentrales Verzeichnis mit diesen Informationen zum Durchsuchen zur Verfügung steht. Die ferne Nachrichtenübertragung zwischen WebSphere MQ-Anwendungen wird im Allgemeinen mithilfe lokaler Definitionen ferner Warteschlangen durchgeführt. Dabei gibt die Anwendung zuerst einen MQOPEN-Aufruf aus, bei dem der Name verwendet wird, der in der lokalen Definition der fernen Warteschlange definiert ist. Um die Nachricht in die ferne Warteschlange einzustellen, gibt die Anwendung eine MQPUT-Anforderung aus, in der die Kennung angegeben ist, die vom MQOPEN-Aufruf zurückgegeben wurde. Die Definition der fernen Warteschlange gibt den Namen der Zielwarteschlange, den Warteschlangenmanager der Zielwarteschlange und optional die Übertragungswarteschlange an. Bei diesem Verfahren muss die Anwendung zur Laufzeit den Namen kennen, der in der lokalen Warteschlangendefinition angegeben ist.

Eine Abweichung zum vorherigen Namen verhindert, dass die lokalen Definitionen ferner Warteschlangen verwendet werden. Die Anwendung kann den vollen Zielwarteschlangennamen angeben, der den Namen

des Warteschlangenmanagers der fernen Warteschlange als Teil von MQOPEN einschließt. Die Anwendung muss deshalb diese beiden Namen zur Laufzeit kennen. Der lokale Warteschlangenmanager muss ordnungsgemäß mit der lokalen Warteschlangendefinition, mit einer entsprechend benannten Übertragungswarteschlange (oder der Standardwarteschlange) sowie mit einem zugehörigen Kanal konfiguriert werden, der Daten an das Ziel zustellt.

Wenn sowohl der Quellen- als auch der Zielwarteschlangenmanager als Member desselben Clusters definiert sind, dann können die Faktoren, die die Übertragungswarteschlange und den Kanal der beiden vorherigen Szenarios betreffen, ignoriert werden. Wenn die Zielübertragungswarteschlange eine Clusterwarteschlange ist, dann ist auch keine lokale Definition einer fernen Warteschlange erforderlich. Ähnlich wie in den zuvor beschriebenen Fällen muss die Anwendung jedoch weiterhin den Namen der Zielwarteschlange kennen.

Ein Verzeichnisservice kann verwendet werden, um diese Anwendungsabhängigkeit in Bezug auf Warteschlangennamen (oder die Kombination aus Warteschlangen- und Warteschlangenmanagernamen) zu beseitigen. Die Zuordnung zwischen Anwendungskriterien und WebSphere MQ-Objektnamen kann in einem Verzeichnis gespeichert und dynamisch und unabhängig von den Anwendungen aktualisiert werden. Zur Laufzeit fragt die WebSphere MQ-Anwendung, die eine Nachricht senden will, das Verzeichnis zuerst mithilfe von anwendungsbasierten Kriterien ab. Hier gilt beispielsweise Folgendes: service_name = "accounts receivable" ruft die relevanten WebSphere MQ-Objektnamen ab und verwendet diese zurückgegebenen Werte dann im MQOPEN-Aufruf.

Ein weiteres Beispiel für die Verwendung eines Verzeichnisses stellt ein Unternehmen dar, das über zahlreiche kleine Lagerhäuser oder Niederlassungen verfügt. Hier können die WebSphere MQ-MQI-Clients zum Senden von Nachrichten an WebSphere MQ-Server verwendet werden, die sich in den größeren Niederlassungen befinden. Die Clients müssen den Namen der Hostmaschine, den MQI-Kanal und den Warteschlangennamen für jeden Server kennen, an den Nachrichten gesendet werden sollen. In bestimmten Fällen kann es notwendig sein, einen WebSphere MQ-Server auf eine andere Maschine zu verschieben. Jeder Client, der mit dem Server kommuniziert, benötigt in diesem Fall die Informationen zur Änderung. Ein LDAP-Verzeichnisservice könnte verwendet werden, um die Namen der Hostmaschinen (sowie die Kanal- und Warteschlangennamen) zu speichern. Die Clientprogramme könnten die Informationen jederzeit aus dem Verzeichnis abrufen, wenn sie eine Nachricht an einen Server senden wollen. In diesem Fall muss nur das Verzeichnis aktualisiert werden, wenn ein Hostname (oder ein Kanal- oder Warteschlangenname) geändert wird.

Mehrere Ziele für eine Anwendungsnachricht können in einem Verzeichnis gespeichert werden. Dabei erfolgt die Auswahl des Ziels abhängig von der Verfügbarkeit und Faktoren der Lastverteilung.

WebSphere MQ kann auch ein LDAP-Verzeichnis verwenden, um die Authentifizierungsdaten zur Verwendung mit SSL (Secure Sockets Layer) zu speichern. WebSphere MQ Classes for Java kann Informationen auch in einem LDAP-Verzeichnis speichern.

LDAP-Beispielprogramm

Das Beispielprogramm ist für Benutzer konzipiert, die bereits über LDAP-Kenntnisse verfügen und eventuell bereits mit dem Produkt arbeiten. Es eignet sich dazu, Ihnen zu zeigen, wie WebSphere MQ-Anwendungen ein LDAP-Verzeichnis nutzen können.

Beispielprogramm erstellen

Dieses Programm wurde ausschließlich unter Windows mit TCP/IP erstellt und getestet. Beachten Sie neben den in „C-Programme unter Windows vorbereiten” auf Seite 487 aufgeführten Hinweisen auch die folgenden Punkte:

- Dieses Programm ist als Clientprogramm konzipiert, sodass es mit der Bibliothek MQIC.LIB verknüpft werden muss.
- Ebenso wie die WebSphere MQ-Headerdateien und -Bibliotheken muss dieses Programm mit LDAP-Client-Headerdateien und -Bibliotheken erstellt werden.

  Verknüpfen Sie das Programm z. B. mithilfe des IBM eNetwork-Clients mit den Bibliotheken LIBLDAPSTATICE.LIB und LIBLBERSTATICSSL.LIB.

Verzeichnis konfigurieren

Vor Ausführung des Beispielprogramms muss ein LDAP-Verzeichnisserver mit Beispieldaten konfiguriert werden.

Die Datei 'MQuser.ldif' im Verzeichnis `tools\c\samples` enthält Beispieldaten in LDIF (LDAP Data Interchange Format). Sie können diese Datei bearbeiten, um Sie an Ihre individuellen Anforderungen anzupassen. Sie enthält Daten für ein fiktives Unternehmen mit dem Namen 'MQuser', dessen Transportabteilung aus drei Niederlassungen besteht. Jede dieser Niederlassungen verfügt über ein System, auf dem ein WebSphere MQ-Server ausgeführt wird.

Sie müssen mindestens die drei Zeilen bearbeiten, die die Hostnamen der Maschinen enthalten, auf denen die WebSphere MQ-Server ausgeführt werden. Dies sind die Zeilen 18, 27 und 36:

```
host: LondonHost
  ...
host: SydneyHost
  ...
host: WashingtonHost
```

Sie müssen anstatt `LondonHost`, `SydneyHost` und `WashingtonHost` die Namen von drei Ihrer eigenen Systeme angeben, auf denen WebSphere MQ-Server ausgeführt werden. Darüber hinaus können Sie die Kanal- und Warteschlangennamen ändern (im Beispiel werden die Systemstandardwerte der Namen verwendet). Außerdem können Sie die Anzahl der Niederlassungen in den Beispieldaten erhöhen oder reduzieren.

IBM Tivoli Directory Server konfigurieren

Informationen zum Installieren des Verzeichnisses finden Sie im Handbuch 'IBM Tivoli Directory Server (ITDS) Administrator's Guide'. Arbeiten Sie im Thema `Installing and Configuring Server`die Abschnitte `Installing Server` und `Basic Server Configuration`durch. Lesen Sie gegebenenfalls den Abschnitt `Administrator Interface` , um sich mit der Funktionsweise der Schnittstelle vertraut zu machen.

Befolgen Sie im Abschnitt `Configuring - How Do I`die Anweisungen zum Start des Administrators. Arbeiten Sie dann den Abschnitt `Configure Database` durch und erstellen Sie eine Standarddatenbank. Überspringen Sie den Abschnitt `Configure replica` und fügen Sie mithilfe des Abschnitts `Work with Suffixes`ein Suffix `o=MQuser`hinzu.

Bevor Sie Einträge zur Datenbank hinzufügen, müssen Sie das Verzeichnisschema erweitern, indem Sie bestimmte Attributdefinitionen und eine Objektklassendefinition hinzufügen. Dies wird im Handbuch IBM Tivoli Directory Server Administrator's Guide im Kapitel `Reference Information` im Abschnitt `Directory Schema`beschrieben. Hierzu werden zwei Beispieldateien bereitgestellt. Die Datei `mq.at.conf` schließt die Attributdefinitionen ein, die Sie der Datei `?etc?slapd.at.conf` hinzufügen müssen. Fügen Sie hierzu die Beispieldatei hinzu, indem Sie die Datei `slapd.at.conf` bearbeiten und folgende Zeile hinzufügen:

```
include <pathname>/mq.at.conf
```

Alternativ hierzu können Sie die Datei `slapd.at.conf` bearbeiten und die Inhalte der Beispieldatei direkt hinzufügen, indem Sie die folgenden Zeilen hinzufügen:

```
# MQ attribute definitions
attribute mqChannel           ces   mqChannel        1000   normal
attribute mqQueueManager      ces   mqQueueManager   1000   normal
attribute mqQueue             ces   mqQueue          1000   normal
attribute mqPort              cis   mqPort           64     normal
```

Ähnlich können Sie für die Objektklassendefinition entweder die Beispieldatei einbinden, indem Sie die Datei `etc?slapd.oc.conf` bearbeiten und die folgende Zeile hinzufügen:

```
include <pathname>/mq.oc.conf
```

Alternativ können Sie den Inhalt der Beispieldatei direkt zu `slapd.oc.conf,` hinzufügen, d. h. die folgenden Zeilen hinzufügen:

```
# MQ object classdefinition
objectclass mqApplication
      requires
              objectClass,
              cn,
              host,
              mqChannel,
              mqQueue
      allows
              mqQueueManager,
              mqPort,
              description,
              l,
              ou,
              seeAlso
```

Nun können Sie den Verzeichnisserver (Verwaltung, Server, Start) starten und die Beispieleinträge hinzufügen. Um die Beispieleinträge hinzuzufügen, rufen Sie 'Administration' (Verwaltung) und dann die Seite 'Add Entries' (Einträge hinzufügen) des Administrators auf. Geben Sie dann den vollständigen Pfadnamen der Beispieldatei `MQuser.ldif` ein und klicken Sie anschließend auf 'Submit' (Übergeben).

Der Verzeichnisserver ist nun aktiv und mit den Daten geladen, die zur Ausführung des Beispielprogramms benötigt werden.

Netscape-Verzeichnisserver konfigurieren

Klicken Sie auf der Seite 'Netscape Server Administration' (Netscape-Serververwaltung) auf **Create New Netscape Directory Server** (Neuen Netscape-Verzeichnisserver erstellen).

Nun wird ein Formular angezeigt, das die Konfigurationsdaten enthält. Ändern Sie das Verzeichnissuffix in **o=MQuser** und fügen Sie ein Kennwort für den uneingeschränkten Benutzer (Unrestricted User) hinzu. Auch die anderen Informationen können geändert werden, um sie an Ihre individuelle Installation anzupassen. Klicken Sie auf **OK**, um das Verzeichnis zu erstellen. Klicken Sie auf **Return to Server Administration** (Zur Serververwaltung zurückkehren) und starten Sie den Verzeichnisserver. Klicken Sie auf den Verzeichnisnamen, um den Directory Server-Verwaltungsserver für das neue Verzeichnis zu starten.

Bevor Sie Einträge zur Datenbank hinzufügen, müssen Sie das Verzeichnisschema erweitern, indem Sie bestimmte Attributdefinitionen und eine Objektklassendefinition hinzufügen. Klicken Sie auf die Registerkarte **Schema** der Directory Server-Seite. Daraufhin wird ein Formular angezeigt, in dem Sie neue Attribute hinzufügen können. Fügen Sie die folgenden Attribute hinzu (geben Sie dabei für die Attribut-OID keinen Wert an):

```
Attribute Name              Syntax
--------------              ------
mqChannel                   Case Exact String
mqQueueManager              Case Exact String
mqQueue                     Case Exact String
mqPort                      Integer
```

Fügen Sie eine neue Objektklasse (objectClass) hinzu, indem Sie in der seitlichen Anzeige auf **Create ObjectClass** (Objektklasse erstellen) klicken. Geben Sie als Objektklassenname **mqApplication** an, wählen Sie als übergeordnete Objektklasse **applicationProcess** aus und lassen Sie das Feld **ObjectClass OID** (OID der Objektklasse) leer. Fügen Sie nun die gewünschten Attribute zur Objektklasse hinzu. Wählen Sie **host**, **mqChannel** und **mqQueue** als erforderliche Attribute und dann **mqQueueManager** und **mqPort** als zulässige Attribute aus. Klicken Sie auf die Schaltfläche **Create New ObjectClass** (Neue Objektklasse erstellen), um die Objektklasse zu erstellen.

Klicken Sie zum Hinzufügen der Beispieldaten auf die Registerkarte **Database Management** (Datenbankmanagement) und wählen Sie dann in der seitlichen Anzeige **Add Entries** (Einträge hinzufügen) aus. Geben Sie den Pfadnamen der Beispieldatendatei `<pathname>\MQuser.ldif` und das Kennwort ein und klicken Sie dann auf **OK**.

Das Beispielprogramm wird als nicht berechtigter Benutzer ausgeführt. Standardmäßig lässt das Netscape-Verzeichnis nicht zu, dass nicht berechtigte Benutzer das Verzeichnis durchsuchen. Um diese Einstellung zu ändern, müssen Sie auf die Registerkarte **Access Control** (Zugriffssteuerung) klicken. Geben Sie das Kennwort des nicht berechtigten Benutzers ein und klicken Sie auf **OK**, um die Zugriffssteuerungseinträge für das Verzeichnis zu laden. Diese Einträge sind momentan leer. Klicken Sie auf die Schaltfläche **New ACI** (Neue ACI), um einen neuen Zugriffssteuerungseintrag zu erstellen. Klicken Sie im angezeigten Eingabefeld auf die (unterstrichene) Option **Deny** (Verweigern) und ändern Sie im daraufhin angezeigten Dialogfeld die Einstellung in **Allow** (Zulassen). Fügen Sie einen Namen wie z. B. **MQuser-access** hinzu und klicken Sie dann auf **choose a suffix** (Suffix auswählen), um **o=MQuser** auszuwählen. Geben Sie **o=MQuser** als Ziel und außerdem das Kennwort des nicht berechtigten Benutzers ein und klicken Sie dann auf **Submit** (Übergeben).

Der Verzeichnisserver ist nun aktiv und mit den Daten geladen, die zur Ausführung des Beispielprogramms benötigt werden.

Beispielprogramm ausführen

Sie verfügen nun über einen aktiven LDAP-Verzeichnisserver, in den die Beispieldaten geladen wurden. In den Daten sind drei Hostmaschinen angegeben, auf denen jeweils ein WebSphere MQ-Server ausgeführt wird. Vergewissern Sie sich, dass der Standardwarteschlangenmanager auf jedem System aktiv ist (sofern Sie nicht die Beispieldaten geändert haben, um einen anderen Warteschlangenmanager anzugeben).

Starten Sie außerdem auf jedem System das WebSphere MQ-Listenerprogramm. Im Beispiel wird TCP/IP mit der WebSphere MQ-Standardportnummer verwendet, sodass Sie den Listener mit dem folgenden Befehl starten können:

```
runmqlsr -t tcp
```

Um das Beispiel zu testen, können Sie außerdem ein Programm ausführen, mit dem die auf den verschiedenen WebSphere MQ-Servern eingehenden Nachrichten gelesen werden. Sie können hierzu z. B. das Beispielprogramm 'amqstrg' verwenden:

```
amqstrg SYSTEM.DEFAULT.LOCAL.QUEUE
```

Das Beispielprogramm verwendet drei Umgebungsvariablen, von denen eine erforderlich und zwei optional sind. Die erforderliche Variable LDAP_BASEDN gibt den Basis-DN für die Verzeichnissuche an. Um mit den Beispieldaten zu arbeiten, müssen Sie hier `ou=Transport, o=MQuser` angeben. Geben Sie an der Eingabeaufforderung auf Windows-Systemen z. B. Folgendes ein:

```
set LDAP_BASEDN=ou=Transport, o=MQuser
```

Die optionalen Variablen sind LDAP_HOST und LDAP_VERSION. Die Variable LDAP_HOST gibt den Namen des Hosts an, auf dem der LDAP-Server ausgeführt wird. Standardmäßig wird der lokale Host verwendet, wenn keine andere Angabe erfolgt. Die Variable LDAP_VERSION gibt die Version des zu verwendenden LDAP-Protokolls an und kann 2 oder 3 sein. Die meisten LDAP-Server unterstützen nun Version 3 des Protokolls; die ältere Version 2 wird von allen LDAP-Servern unterstützt. Dieses Beispiel funktioniert mit beiden Protokollversionen gleich gut und es wird standardmäßig Version 2 verwendet, falls nichts anderes angegeben ist.

Sie können nun das Beispiel ausführen, indem Sie den Programmnamen gefolgt vom Namen der WebSphere MQ-Anwendung angeben, an die Nachrichten gesendet werden sollen. In den Beispieldaten lau-

ten die Anwendungsnamen 'London', 'Sydney' und 'Washington'. Gehen Sie wie folgt vor, um Nachrichten an die Anwendung 'London' zu senden:

```
amqsldpc London
```

Wenn das Programm keine Verbindung zum WebSphere MQ-Server herstellen kann, wird eine entsprechende Fehlernachricht ausgegeben. Wenn die Verbindung erfolgreich hergestellt werden kann, können Sie mit der Eingabe von Nachrichten beginnen. Jede Zeile, die Sie eingeben (durch < return> oder < enter> beendet), wird als separate Nachricht gesendet, eine leere Zeile beendet das Programm.

Programmentwurf

Das Programm verfügt über zwei separate Komponenten. Die erste Komponente verwendet die Umgebungsvariablen und den Befehlszeilenwert zum Abfragen eines LDAP-Verzeichnisservers, die zweite richtet die WebSphere MQ-Verbindung mit Informationen ein, die vom Verzeichnis zurückgegeben wurden, und sendet die Nachrichten.

Die LDAP-Aufrufe, die in der ersten Komponente des Programms verwendet werden, können abhängig davon, ob LDAP Version 2 oder 3 verwendet wird, geringfügig variieren. Sie werden detailliert in der Dokumentation beschrieben, die zum Lieferumfang der LDAP-Clientbibliotheken gehört. Der vorliegende Abschnitt enthält eine kurze Beschreibung.

Die erste Programmkomponente überprüft, ob es ordnungsgemäß aufgerufen wurde und liest die Umgebungsvariablen. Sie stellt dann eine Verbindung zum LDAP-Verzeichnisserver auf dem angegebenen Host her:

```
if (ldapVersion == LDAP_VERSION3)
{
  if ((ld = ldap_init(ldapHost, LDAP_PORT)) == NULL)
     ...
}
else
{
  if ((ld = ldap_open(ldapHost, LDAP_PORT)) == NULL )
     ...
}
```

Wurde eine Verbindung hergestellt, legt das Programm mit dem Aufruf "ldap_set_option" bestimmte Optionen auf dem Server fest. Anschließend authentifiziert sich das Programm beim Server, indem es wie folgt eine Bindung herstellt:

```
if (ldapVersion == LDAP_VERSION3)
{
  if (ldap_simple_bind_s(ld, bindDN, password) != LDAP_SUCCESS)
     ...
}
else
{
  if (ldap_bind_s(ld, bindDN, password, LDAP_AUTH_SIMPLE) !=
      LDAP_SUCCESS)
     ...
}
```

Im Beispielprogramm werden `bindDN` und `password` auf NULL gesetzt. Dies bedeutet, dass das Programm sich als anonymer Benutzer authentifiziert und somit über keine speziellen Zugriffsberechtigungen verfügt und nur auf Informationen zugreifen kann, die öffentlich verfügbar sind. In der Praxis beschränken die meisten Unternehmen den Zugriff auf die Informationen, die in Verzeichnissen gespeichert werden, sodass nur autorisierte Benutzer darauf zugreifen können.

Der erste Parameter für den Bindungsaufruf ist `ld`. Hierbei handelt es sich um eine Kennung, die verwendet wird, um diese spezielle LDAP-Sitzung im restlichen Programm zu identifizieren. Nach der Authenti-

fizierung durchsucht das Programm das Verzeichnis nach Einträgen, die mit dem Anwendungsnamen übereinstimmen:

```
rc = ldap_search_s(ld,                /* LDAP Handle             */
                   baseDN,            /* base distinguished name */
                   LDAP_SCOPE_ONELEVEL, /* one-level search       */
                   filterPattern,     /* filter search pattern   */
                   attrs,             /* attributes required     */
                   FALSE,             /* NOT attributes only     */
                   &ldapResult);      /* search result           */
```

Dies ist ein einfacher synchroner Aufruf an den Server, der die Ergebnisse direkt zurückgibt. Es stehen weitere Suchtypen zur Verfügung, die sich besser für komplexe Abfragen oder für Fälle eignen, bei denen eine große Anzahl von Ergebnissen zu erwarten ist. Der erste Parameter für die Suche ist die Kennung `ld`, die die Sitzung identifiziert. Der zweite Parameter ist der Basis-DN, in dem angegeben ist, an welcher Position innerhalb des Verzeichnisses mit der Suche begonnen werden soll. Der dritte Parameter gibt den Bereich der Suche und damit die Einträge an, die relativ zum Ausgangspunkt gesucht werden sollen. Diese beiden Parameter definieren gemeinsam, welche Einträge im Verzeichnis gesucht werden. Der nächste Parameter (`filterPattern`) gibt an, wonach gesucht werden soll. Der Parameter `attrs` listet die Attribute auf, die aus dem Objekt zurückgegeben werden sollen, nachdem es gefunden wurde. Das nächste Attribut gibt an, ob lediglich die Attribute oder auch ihre Werte zurückgegeben werden sollen. Wenn Sie hier FALSE angeben, dann werden auch die Attributwerte zurückgegeben. Der letzte Parameter wird verwendet, um das Ergebnis zurückzugeben.

Das Ergebnis kann viele Verzeichniseinträge enthalten, die jeweils über die angegebenen Attribute und Werte verfügen. Aus dem Ergebnis müssen die gewünschten Werte extrahiert werden. Im vorliegenden Beispielprogramm wird erwartet, dass lediglich ein Eintrag gefunden wird. Aus diesem Grund wird nur der erste Eintrag im Ergebnis geprüft:

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

Dieser Aufruf gibt eine Kennung für den ersten Eintrag zurück. Es wird eine For-Schleife eingerichtet, um alle Attribute aus dem Eintrag zu extrahieren:

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);
     attribute != NULL;
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))
{
```

Für jedes dieser Attribute werden die zugeordneten Werte extrahiert. Auch in diesem Fall wird nur ein Wert pro Attribut erwartet, sodass nur der erste Wert verwendet wird. Es wird ermittelt, welches Attribut vorhanden ist, und der Wert wird in der entsprechenden Programmvariablen gespeichert:

```
values = ldap_get_values(ld, ldapEntry, attribute);
if (values != NULL && values[0] != NULL)
{
  if (stricmp(attribute, MQ_HOST_ATTR) == 0)
  {
    mqHost = strdup(values[0]);
    ...
```

Abschließend wird der belegte Speicher freigegeben (ldap_value_free, ldap_memfree, ldap_msgfree) und die Sitzung wird geschlossen, indem die *Bindung zum Server aufgehoben* wird:

```
ldap_unbind(ld);
```

Anschließend wird überprüft, ob alle WebSphere MQ-Werte gefunden wurden, die aus dem Verzeichnis benötigt werden. In diesem Fall wird sendMessages() aufgerufen, um eine Verbindung zum WebSphere MQ-Server herzustellen und die WebSphere MQ-Nachrichten zu senden.

Die zweite Komponente des Beispielprogramms ist die Routine sendMessages(), die alle WebSphere MQ-Aufrufe enthält. Sie wird im Beispielprogramm 'amqsput0' modelliert. Der Unterschied besteht darin, dass die Parameter aus dem Programm erweitert wurden und anstelle des MQCONN-Aufrufs MQCONNX verwendet wird.

Anwendungen für IBM WebSphere MQ Telemetry entwickeln

Telemetrieanwendungen integrieren Sensor- und Steuerungsgeräte mit anderen Informationsquellen, die im Internet oder im Unternehmen zur Verfügung stehen.

Es können Anwendungen für IBM WebSphere MQ Telemetry mithilfe von Designmustern, Arbeitsbeispielen, Beispielprogrammen, Programmierungskonzepten und Referenzinformationen entwickelt werden. Verwenden Sie den IBM WebSphere MQ Telemetry-Dämon für Einheiten, um das Verbinden vieler kleiner Einheiten mit IBM WebSphere MQ zu erleichtern.

Zugehörige Konzepte

WebSphere MQ Telemetry

Telemetriekonzepte und -szenarios im Bereich der Überwachung und Steuerung

Zugehörige Tasks

WebSphere MQ Telemetry installieren

WebSphere MQ Telemetry verwalten

Fehlerbehebung für WebSphere MQ Telemetry

Zugehörige Verweise

WebSphere MQ Telemetry-Referenz

IBM WebSphere MQ Telemetry Beispielprogramme

Um die grundlegende Verwendung der MQ Telemetry Transport V3-Clientanwendung zu veranschaulichen, werden Beispielscripts bereitgestellt. Verwenden Sie die Scripts, um eine Nachricht für ein Thema zu veröffentlichen und ein Thema zu subskribieren.

Vorbereitende Schritte

Starten Sie den Telemetry-Service (MQXR), um die Beispielprogramme auszuführen.

Die Benutzer-ID muss Mitglied der Benutzergruppe 'mqm' sein.

Führen Sie zuerst das Script 'SampleMQM' und dann das Script 'MQTTV3Sample' aus, um eine Veröffentlichung und eine Subskription durchzuführen. Führen Sie das Beispielscript 'CleanupMQM' aus, um den mit dem Script 'SampleMQM' erstellten Warteschlangenmanager zu löschen.

Da das Script 'SampleMQM' einen Warteschlangenmanager mit dem Namen QM1 erstellt und verwendet, darf es nicht unverändert auf einem System ausgeführt werden, auf dem es bereits einen Warteschlangenmanager QM1 gibt. Jegliche Änderungen können Auswirkungen auf die Konfiguration des vorhandenen Warteschlangenmanagers haben.

Informationen zu diesem Vorgang

- Die Anwendung 'SampleMQM' erstellt und startet einen für die Telemetrie aktivierten Warteschlangenmanager namens QM1. Das Script richtet außerdem eine Standardübertragungswarteschlange für QM1 ein und erstellt und startet einen Standardkanal, der am Port 1883 empfangsbereit ist. Dieser Kanal führt keine Authentifizierung von mit ihm verbundenen Clients durch. Der Kanal verfügt über das Attribut MCAUSER (Message Channel Agent User Identifier), das auf Windows -Systemen auf 'guest' und auf Linux -Systemen auf 'nobody' gesetzt ist. Mit dem Kanal verbundene Clients werden je nach Betriebssystem wie der Benutzer 'guest' oder der Benutzer 'nobody' behandelt. Das Script berechtigt 'guest' auf Windows -Systemen und 'nobody ' auf Linux -Systemen zur Veröffentlichung und Subskription eines Themas auf QM1 .
- Das Script 'MQTTV3Sample' befindet sich an der folgenden Position:

– Unter Windows *MQ_INSTALLATION_PATH*\mqxr\samples

  Dabei steht *MQ-INSTALLATIONSPFAD* für die Position, an der IBM WebSphere MQ installiert ist.

– Unter Linux *MQ_INSTALLATION_PATH*/mqxr/samples

Die Anwendung 'MQTTV3Sample' fungiert als Publisher, der eine einzelne Nachricht zu einem Thema auf dem Server sendet. Sie fungiert außerdem als Subskribent, der für Nachrichten vom Server empfangsbereit ist.

- Das Beispielscript 'CleanupMQM' beendet und löscht QM1, der vom Script 'SampleMQM' erstellt wurde. Verwenden Sie das Beispielscript 'CleanupMQM', wenn Sie das Script 'SampleMQM' erneut ausführen oder QM1 entfernen möchten.

Vorgehensweise

1. Geben Sie den folgenden Befehl in einer Befehlszeile ein, um das Script 'SampleMQM' auszuführen.
   - Unter Windows lautet der Befehl zur Ausführung des Scripts 'SampleMQM' wie folgt:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\SampleMQM.bat
     ```

   - Unter AIX und Linux lautet der Befehl zur Ausführung des Scripts 'SampleMQM' wie folgt:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

   Dabei steht *MQ-INSTALLATIONSPFAD* für die Position, an der IBM WebSphere MQ installiert ist.

   Ein Warteschlangenmanager mit dem Namen MQXR_SAMPLE_QM wird erstellt.
2. Geben Sie den folgenden Befehl ein, um den ersten Teil des Scripts 'MQTTV3Sample' auszuführen.
   - Geben Sie unter Windows in einer Befehlszeile folgenden Befehl ein:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -a subscribe
     ```

   - Geben Sie unter AIX und Linuxin einem Shellfenster den folgenden Befehl ein:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscribe
     ```

3. Geben Sie den folgenden Befehl ein, um den zweiten Teil des Scripts 'MQTTV3Sample' auszuführen.
   - Geben Sie unter Windows in einer anderen Befehlszeile folgenden Befehl ein:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -m "Hello from an MQTT v3 applicati
     on"
     ```

   - Geben Sie unter AIX und Linuxin einem anderen Shellfenster den folgenden Befehl ein.

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -m "Hello from an MQTT v3 application"
     ```

4. Wenn Sie den Warteschlangenmanager, der vom Script 'SampleMQM' erstellt wurde, entfernen möchten, können Sie das Script 'CleanupMQM' mit folgendem Befehl ausführen.
   - Geben Sie unter Windows folgenden Befehl ein:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat
     ```

   - Geben Sie unter AIX und Linux in einem anderen Shellfenster den folgenden Befehl ein.

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh
     ```

Ergebnisse

Die Nachricht `Hello from an MQTT v3 application`, die Sie im zweiten Fenster eingegeben haben, wird von dieser Anwendung veröffentlicht und von der Anwendung im ersten Fenster empfangen. Die Anwendung im ersten Fenster zeigt sie dann auf dem Bildschirm an.

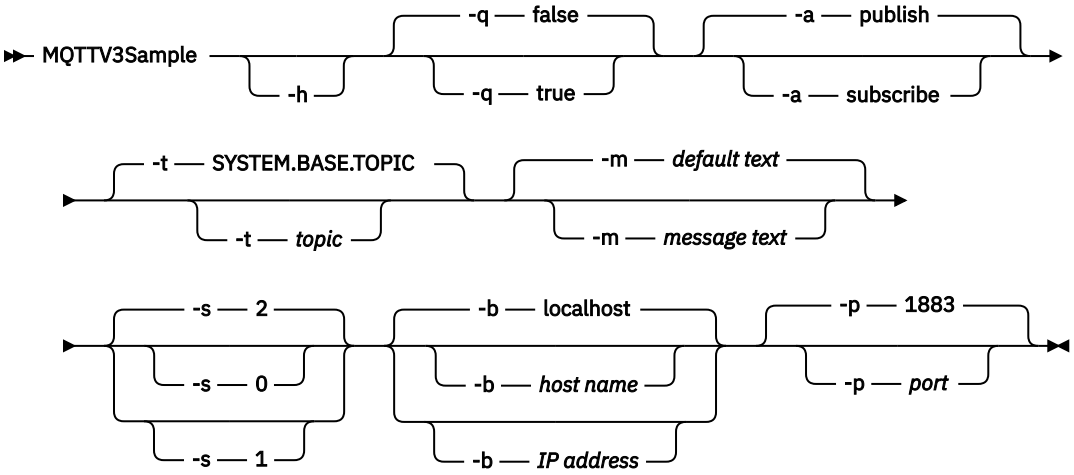
Programm MQTTV3Sample

Referenzinformationen zur Beispielsyntax und zu Parametern für das Programm 'MQTTV3Sample'.

Verwendungszweck

Mit dem Programm 'MQTTV3Sample' kann eine Nachricht für ein Thema veröffentlicht und subskribiert werden.

MQTTV3Sample syntax



Parameter

-h
Der Hilfetext wird ausgedruckt und das Script wird beendet.

-q
Einstellen des Standardmodus 'false' anstatt des Befehlszeilenmodus.

-a
Angabe von 'publish' oder 'subscribe' statt Übernahme der Veröffentlichung als Standardaktion.

-t
Veröffentlichen oder subskribieren eines Themas, statt das Standardthema zu veröffentlichen oder zu subskribieren.

-m
Veröffentlichen von Nachrichtentext statt Versand des standardmäßigen Veröffentlichungstextes "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

Verwenden Sie zum Subskribieren eines Themas unter Windows den folgenden Befehl:

```
runMQTTV3Sample -a subscribe
```

Verwenden Sie zum Veröffentlichen einer Nachricht unter Windows den folgenden Befehl:

```
runMQTTV3Sample
```

Weitere Informationen zur Ausführung des bereitgestellten Beispielscripts finden Sie im Abschnitt „IBM WebSphere MQ Telemetry Beispielprogramme” auf Seite 502.

Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen

Die zur Erstellung einer MQTT-Clientanwendung erforderlichen Schritte werden in Form eines Lernprogramms beschrieben. Es wird jede Code-Zeile erklärt. Am Ende der Task haben Sie einen MQTT-Publisher erstellt. Die Veröffentlichungen können Sie mit WebSphere MQ Explorer durchsuchen.

Vorbereitende Schritte

Installieren Sie das Feature WebSphere MQ Telemetry auf einem Server, auf dem IBM WebSphere MQ Version 7.1 oder höher installiert ist.

Die Clientanwendung verwendet das `com.ibm.mq.micro.client.mqttv3`-Paket im IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Das SDK ist Bestandteil der IBM WebSphere MQ Telemetry-Installation. Der Client stellt eine Verbindung zu der IBM WebSphere MQ Telemetry-Funktion her, um Nachrichten mit IBM WebSphere MQ auszutauschen.

Sie müssen außerdem die Telemetrie-Aktualisierungen für IBM WebSphere MQ Explorer Version 7.1 installieren, um IBM WebSphere MQ Telemetry zu verwalten. Die Updates sind Bestandteil der IBM WebSphere MQ Telemetry-Installation.

Ein MQTT-Client, der in Java SE ausgeführt wird, erfordert Version 6.0 von Java SE oder höher. IBM Java SE v6.0 ist Teil der IBM WebSphere MQ Version 7.1 -Installation. Sie befindet sich unter *WebSphere MQ installation directory*`\java\jre` .

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Veröffentlichungsanwendung, PubSync. PubSync veröffentlicht `Hello World` zum Thema `MQTT Examples`und wartet auf die Bestätigung, dass die Veröffentlichung an den Warteschlangenmanager zugestellt wurde.

Durch Einrichtung einer permanenten Subskription für `MQTT Examples` können Sie überprüfen, ob die Anwendung funktioniert.

Die Prozedur verwendet Eclipse für die Entwicklung, Erstellung und Ausführung des Clients. Sie können Eclipse von der Website des Eclipse-Projekts unter www.eclipse.org herunterladen.

Zur Erstellung der Anwendung können Sie die Java-Dateien erstellen und sie über die Befehlszeile kompilieren und ausführen.

Erstellen Sie in einem neuen Verzeichnis den Verzeichnispfad `.\com\ibm\mq\id`. Erstellen Sie zwei Java-Dateien: `Example.java` und `PubSync.java`. Kopieren Sie den Code aus „Beispielcode” auf Seite 509 in die Java-Dateien.

Kompilieren Sie den Java-Code mit folgendem Befehl:

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubSync.java com.ibm.mq.id.Example.java
```

Führen Sie PubSync mithilfe des folgenden Befehls aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubSync
```

Vorgehensweise

1. Erstellen Sie ein Java-Projekt in Eclipse.

a) **Datei > Neu > Java-Projekt** und geben Sie einen Projektnamen ein. Klicken Sie auf **Weiter**.

Prüfen Sie, ob die korrekte JRE-Version oder eine höhere Version verwendet wird. Java SE muss 6.0 oder höher aufweisen.

b) Klicken Sie auf der Seite 'Java-Einstellungen' auf **Bibliotheken > Externe JAR-Dateien hinzufügen ...** .

c) Navigieren Sie zu dem Verzeichnis, in dem Sie den Ordner des WebSphere MQ Telemetry-Software-Development-Kits installiert haben. Suchen Sie den Ordner `SDK\clients\java`, wählen Sie alle JAR-Dateien (`.jar`) aus und klicken Sie dann auf **Öffnen > Fertigstellen**.

2. Installieren Sie die Javadoc für den MQTT-Client.

Wenn die Javadoc für den MQTT-Client installiert ist, bietet der Java-Editor Unterstützung mit MQTT v3-Klassen.

a) Öffnen Sie in Ihrem Java-Projekt **Paketexplorer > Referenzierte Bibliotheken**. Klicken Sie mit der rechten Maustaste auf `com.ibm.micro.client.mqttv3.jar` und dann auf **Eigenschaften**.

b) Klicken Sie im Eigenschaftennavigator auf **Javadoc Location (Javadoc-Position)**.

c) Klicken Sie auf der Seite 'Javadoc-Position' auf **Javadoc-URL > Durchsuchen ...** . und suchen Sie den Ordner *`WMQ Installation directory`*`\mqxr\SDK\clients\java\doc\javadoc` **> OK**.

d) Klicken Sie auf **Validieren > OK**.

Sie werden aufgefordert, einen Browser zum Anzeigen der Dokumentation zu öffnen.

3. Erstellen Sie die Klasse PubSyncmit dem Assistenten für Java-Klassen.

a) Klicken Sie mit der rechten Maustaste auf das von Ihnen erstellte Java-Projekt > **Neu** > **Klasse**.

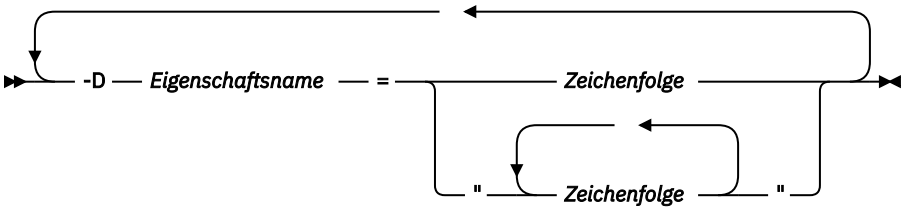
b) Geben Sie den Paketnamen `com.ibm.mq.id` ein.

c) Geben Sie den Klassennamen PubSync ein.

d) Aktivieren Sie das Methodenstub-Feld **public static void main(String [] args)**

4. Erstellen Sie die Datei `Example.java` im Paket `com.ibm.mq.id`. Kopieren Sie den Code aus Abbildung 89 auf Seite 511 in die Datei.

Alle in den Beispielen verwendete Parameter werden als Eigenschaften festgelegt. Sie können die Werte überschreiben, indem Sie die Standardeinstellungen in `Example.java`ändern oder die Eigenschaften als Optionen in der Java-Befehlszeile mit dem Parameter `-D` angeben:



Die in diesem Beispiel und in den Beispielen des Abschnitts „Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen" auf Seite 511 verwendete Client-ID ist ein Benutzername, dem als Suffix eine Zufallszeichenfolge angefügt ist.

5. Führen Sie die Schritte zum Erstellen des Codes aus oder kopieren Sie den Code aus dem Abschnitt Abbildung 88 auf Seite 510.

In den folgenden Schritten wird der Code in `Pubsync.java` erläutert.

6. Erstellen Sie einen `try-catch`-Block.

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

Der MQTT-Client gibt die Ausnahme `MqttException`, `MqttPersistenceException` oder `MqttSecurityException` aus. `MqttPersistenceException` und `MqttSecurityException` sind Unterklassen von `MqttException`.

Verwenden Sie die Methode `MqttException.getReasonCode`, um die Ursache der Ausnahme zu ermitteln. Verwenden Sie bei Ausgabe von `MqttPersistenceException` oder `MqttSecurityException` die Methode `getCause`, um die zugrundeliegende Throwable-Ausnahme zurückzugeben.

7. Erstellen Sie eine neue `MqttClient`-Instanz.

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

Stellen Sie dem Client eine Serveradresse zur Verfügung, die später für die Verbindung mit WebSphere MQ verwendet werden soll. Legen Sie die Client-ID zur Benennung des Clients fest.

- Optional können Sie eine Implementierung der Schnittstelle `MqttClientPersistence` bereitstellen, durch die die Standardschnittstelle ersetzt wird. Die Standardimplementierung `MqttPersistence` speichert QoS 1 und 2 Nachrichten, die auf die Zustellung warten, als Dateien; siehe „Nachrichtenpersistenz in MQTT-Clients” auf Seite 566.
- Der standardmäßige IBM WebSphere MQ-TCP/IP-Port für MQTT lautet 1883. Für SSL ist es Port 8883. Im Beispiel wird die Standardadresse auf `tcp://localhost:1883` gesetzt.
- Normalerweise muss ein bestimmter physischer Client anhand seiner Client-ID eindeutig zu erkennen sein. Die Client-ID muss daher für alle Clients, die eine Verbindung zu einem Server herstellen, eindeutig sein (siehe „Client-ID” auf Seite 561). Wenn für eine Instanz die gleiche Client-ID wie für eine frühere Instanz verwendet wird, bedeutet dies, dass die aktuelle Instanz eine Instanz des selben Clients ist. Wird eine Client-ID auf zwei aktiven Clients doppelt vergeben, wird auf beiden Clients eine Ausnahme ausgelöst und ein Client wird beendet.
- Die Länge der Client-ID ist auf 23 Bytes beschränkt. Bei Überschreiten dieser Länge wird eine Ausnahme ausgegeben. Außerdem darf die Client-ID nur Zeichen enthalten, die auch in den Namen von Warteschlangenmanagern erlaubt sind. Bindestriche und Leerzeichen sind zum Beispiel nicht zulässig.
- Eine Nachrichtenverarbeitung erfolgt erst, wenn Sie die Methode `MqttClient.connect` aufgerufen haben.

Verwenden Sie das Clientobjekt zum Veröffentlichen und Subskribieren von Themen und zum Wiederherstellen von Informationen zu Veröffentlichungen, die noch nicht zugestellt wurden.

8. Erstellen Sie ein Thema, in dem Veröffentlichungen vorgenommen werden können.

```
 MqttTopic topic = client.getTopic(Example.topicString);
```

Themenzeichenfolgen sind auf 64 KB beschränkt. Dies ist länger als für Themenzeichenfolgen in IBM WebSphere MQ zulässig ist. Abgesehen von dieser Ausnahme gelten für Themenzeichenfolgen die gleichen Regeln wie in WebSphere MQ (siehe Themenzeichenfolgen). In diesem Beispiel wird die Topic-Zeichenfolge `MQTT Examples` festgelegt.

9. Erstellen Sie eine Veröffentlichungsnachricht.

```
MqttMessage message = new MqttMessage(Example.publication.getBytes());
```

Die Zeichenfolge `"Hello World"` wird in eine Bytefeldgruppe konvertiert und zur Erstellung einer MQTT-Nachricht (`MqttMessage`) verwendet.

- Die Nutzdaten einer MQTT-Nachricht liegen immer in Form einer Bytefeldgruppe vor. Die Methode `getBytes` konvertiert ein Zeichenfolgeobjekt in UTF-8. Die Nachricht `MqttMessage` enthält eine `toString`-Methode, mit der die Nachrichtennutzdaten als Zeichenfolge zurückgegeben werden. Sie ist äquivalent zu `new string(message.getPayload)`.
- An den Warteschlangenmanager wird eine Veröffentlichungsnachricht mit einem RFH2-Header gesendet und die Nachrichtendaten werden als `jms-bytes`-Nachricht gesendet.
- Das Nachrichtenobjekt verfügt über die Attribute "Servicequalität" und "Ständige Veröffentlichung". Die Servicequalität (QoS) bestimmt die Zuverlässigkeit der Nachrichtenübertragung zwischen dem

MQTT-Client und dem Warteschlangenmanager; siehe „Von einem MQTT-Client bereitgestellte Servicequalität” auf Seite 571. Das Attribut "Ständige Veröffentlichung" legt fest, ob eine Veröffentlichung vom Warteschlangenmanager für künftige Subskribenten gespeichert wird. Nicht ständige Veröffentlichungen werden nur an aktuelle Subskribenten gesendet (siehe „Ständige Veröffentlichungen und MQTT-Clients” auf Seite 573). Wenn die `MqttMessage`-Standardeinstellungen übernommen werden, gilt: "Nachrichten werden mindestens einmal übermittelt, aber nicht beibehalten."

10. Stellen Sie eine Verbindung zum Server her.

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

In diesem Beispiel wird die Verbindung zum Server mit den Standardverbindungseinstellungen hergestellt. Sobald die Verbindung hergestellt ist, können Sie mit der Veröffentlichung beginnen. Die Standardverbindungsoptionen lauten wie folgt:

- Alle 15 Sekunden wird eine kurze Keepalive-Nachricht gesendet, damit die TCP/IP-Verbindung nicht geschlossen wird.
- Die Sitzung wird ohne Überprüfung, ob vorangegangene Veröffentlichungen abgeschlossen sind, gestartet.
- Das Intervall zwischen aufeinanderfolgenden Versuchen, eine Nachricht zu übertragen, beträgt 15 Sekunden.
- Für die Verbindung wird keine Nachricht 'Last Will and Testament' erstellt.
- Zur Herstellung der Verbindung wird die Standard-Socket-Factory verwendet.

Wenn Sie die Verbindungsoptionen ändern möchten, erstellen Sie das Objekt `ConnectionOptions` und übergeben es als zusätzlichen Parameter an `client.connect`.

11. Veröffentlichen.

```
MqttDeliveryToken token = topic.publish(message);
```

In diesem Beispiel wird die Veröffentlichung "Hello World" zum Thema "MQTT Examples" an den Warteschlangenmanager gesendet.

- Wenn die Rückgabe der Methode `publish` eintrifft, wurde die Nachricht erfolgreich an den MQTT-Client gesendet, jedoch noch nicht an den Server übertragen. Hat die Nachricht die Servicequalität QoS1 oder 2, wird sie lokal gespeichert, für den Fall, dass es auf dem Client zu einem Fehler kommt, bevor die Zustellung abgeschlossen ist.
- `publish` gibt ein Zustellungstoken zurück, anhand dem überprüft wird, ob vom Server bereits eine Bestätigung eingetroffen ist.

12. Warten Sie auf die Empfangsbestätigung des Servers.

```
token.waitForCompletion(Example.timeout);
```

Im Beispiel 'PubSync' wird auf eine Empfangsbestätigung des Servers gewartet, mit der die Nachrichtenzustellung bestätigt wird.

- Ohne Zeitlimit würde der Client bei Nichteintreffen der Bestätigung unbegrenzt warten. In der Task „Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen” auf Seite 511 wird gezeigt, wie Bestätigungen mittels eines Callback-Objekts auch ohne Warten empfangen werden können.

13. Trennen Sie die Verbindung des Clients zum Server.

```
client.disconnect();
```

Der Client trennt die Verbindung zum Server und wartet auf `MqttCallback`-Methoden, die eventuell noch zur Fertigstellung des Vorgangs ausgeführt werden. Danach wartet er noch maximal 30 Sekunden, um eventuell verbleibende Aufgaben fertigzustellen. Sie können als zusätzlichen Parameter ein Quiesce-Zeitlimit festlegen.

14. Speichern Sie Änderungen an den Dateien PubSync.java und Example.Java.

    Eclipse kompiliert automatisch den Java-Code. Wenn Sie das Programm ausführen, können Sie nun die Ergebnisse anzeigen.

Ergebnisse

Wenn Sie die Veröffentlichungen unter Verwendung von WebSphere MQ anzeigen möchten, erstellen Sie ein Thema, eine Warteschlange und eine permanente Subskription, die alle als "MQTTExampleTopic" bezeichnet werden. Hierfür wird das Script in Abbildung 87 auf Seite 509 verwendet. Führen Sie den Client für die Veröffentlichung zum Thema MQTT Examples aus und führen Sie dann das Beispielprogramm **amqsbcg** aus, um die Veröffentlichungen in der MQTTExamples -Warteschlange zu durchsuchen.

1. Starten Sie einen Warteschlangenmanager und danach den zugehörigen Telemetrieservice (MQXR). Stellen Sie sicher, dass die TCP/IP-Adresse und der Port, die für den Telemetriekanal konfiguriert sind, den in der MQTT-Anwendung verwendeten Werten entsprechen.
2. Konfigurieren Sie eine permanente Subskription, indem Sie das Befehlsscript mqttexamples.txt erstellen und mit **runmqsc:** ausführen.

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Abbildung 87. mqttExampleTopic.txt

   Zur Ausführung des Scripts unter Windows geben Sie folgenden Befehl ein:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Führen Sie den Client als Java-Anwendung in Eclipseoder durch Ausführen von Java in einem Befehlsfenster aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

   Anmerkung: Das Befehlsfenster muss in dem Verzeichnis geöffnet sein, in dem sich der Pfad befindet com\ibm\mq\id.
4. Zeigen Sie die Ergebnisse entweder mit WebSphere MQ Explorer oder durch Ausführung des folgenden Befehls an:

```
amqsbcg MQTTExampleQueue queue manager name
```

Beispielcode

In PubSync.java ist der gesamte im Abschnitt Vorgehensweise beschriebene Code aufgeführt. Ändern Sie die Klasse Example in Abbildung 89 auf Seite 511 , um die in PubSync.java verwendeten Standardparameter zu überschreiben.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      client.connect();
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
          + " seconds for publication of \"" + message.toString()
          + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName()
          + "\" for client instance: \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 88. PubSync.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Abbildung 89. Example.java

Zugehörige Konzepte

MQTT-Publish/Subscribe-Anwendungen

Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen

Bei dieser Task führen Sie ein Lernprogramm aus, um Ihre erste Publisher-Anwendung zu ändern. Mit den Änderungen kann die Anwendung Veröffentlichungen senden, ohne auf Empfangsbestätigungen warten zu müssen. Die Empfangsbestätigungen werden von einer von Ihnen zu erstellenden Callback-Klasse empfangen.

Vorbereitende Schritte

Installieren Sie das Feature WebSphere MQ Telemetry auf einem Server, auf dem IBM WebSphere MQ Version 7.1 oder höher installiert ist.

Die Clientanwendung verwendet das `com.ibm.mq.micro.client.mqttv3`-Paket im IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Das SDK ist Bestandteil der IBM WebSphere MQ Telemetry-Installation. Der Client stellt eine Verbindung zu der IBM WebSphere MQ Telemetry-Funktion her, um Nachrichten mit IBM WebSphere MQ auszutauschen.

Sie müssen außerdem die Telemetrie-Aktualisierungen für IBM WebSphere MQ Explorer Version 7.1 installieren, um IBM WebSphere MQ Telemetry zu verwalten. Die Updates sind Bestandteil der IBM WebSphere MQ Telemetry-Installation.

Ein MQTT-Client, der in Java SE ausgeführt wird, erfordert Version 6.0 von Java SE oder höher. IBM Java SE v6.0 ist Teil der IBM WebSphere MQ Version 7.1 -Installation. Sie befindet sich unter *WebSphere MQ installation directory*\java\jre .

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Veröffentlichungsanwendung,PubAsync. PubAsync veröffentlicht `Hello World` zum Thema `MQTT Examples`, ohne auf die Bestätigung zu warten, dass die Veröffentlichung an den Warteschlangenmanager zugestellt wurde. Die Empfangsbestätigungen werden in einer Callback-Klasse (`CallBack`) empfangen.

Durch Einrichtung einer permanenten Subskription für `MQTT Examples` können Sie überprüfen, ob die Anwendung funktioniert.

Die Prozedur verwendet Eclipse für die Entwicklung, Erstellung und Ausführung des Clients. Sie können Eclipse von der Website des Eclipse-Projekts unter www.eclipse.org herunterladen.

Mit den unter Vorgehensweise erläuterten Schritten wird die Anwendung PubSync`.java` aus dem Abschnitt „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 geändert.

Alternativ können Sie den Code unter „Beispielcode” auf Seite 514 in ein neues Verzeichnis (`.\com\ibm\mq\id`) kopieren. Erstellen Sie drei Java-Dateien: `Example.java`, `CallBack.javaund` PubAsync`.java`. Kompilieren Sie die Beispiele mit folgendem Befehl:

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsync.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Example.java
```

Führen Sie PubAsync mit folgendem Befehl aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsync.class
```

Vorgehensweise

1. Erstellen Sie im Paket `com.ibm.mq.id` eine Datei, `CallBack.java`. Kopieren Sie den Code aus Abbildung 92 auf Seite 515 in die Datei.

   Mit `Callback.java` wird die `MqttCallBack`-Schnittstelle implementiert. Im Beispiel initialisiert ein zusätzlicher Konstruktor den Callback mit einigen Instanzdaten.
2. Klicken Sie im Paket `com.ibm.mq.id` mit der rechten Maustaste auf die Datei PubSync`.java` und kopieren Sie sie. Fügen Sie die Datei in dasselbe Paket ein und benennen Sie sie dabei in PubAsync um.
3. Erstellen Sie direkt vor der Codezeile `client.connect();` eine Instanz der Klasse `CallBack` unter Angabe der Client-ID.

   ```
   CallBack callback = new CallBack(Example.clientId);
   client.setCallback(callback);
   ```

   - Die Klasse `CallBack` implementiert `MqttCallBack`. Es ist eine Callback-Instanz pro Client-ID erforderlich. In diesem Beispiel übergibt der Konstruktor die zu speichernde Client-ID als Instanzdaten. Mit ihrer Hilfe wird im Callback ermittelt, welche Instanz des Callbacks gestartet wurde.
   - Sie müssen drei Methoden in der Callback-Klasse implementieren:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Empfängt eine Veröffentlichung, die subskribiert wurde.

     `public void connectionLost(Throwable cause)`
     Wird aufgerufen, wenn die Verbindung verloren geht.

public void deliveryComplete(MqttDeliveryToken token))
Wird aufgerufen, wenn ein Zustellungstoken für eine Nachricht mit Servicequalität 1 oder 2, die veröffentlicht wurde, empfangen wird.

- Der Callback wird durch `MqttClient.connect` aktiviert.

4. Trennen Sie die Verbindung zum Client.
   a) Entfernen Sie die Anweisung mit dem Ausdruck `token.waitForCompletion`.
   Der Hauptthread wird fortgesetzt, ohne dass auf die Zustellung der Veröffentlichung gewartet wird.
   b) Testen Sie, ob die Clientverbindung bereits getrennt ist.
   Die Verbindung des MQTT-Clients wird getrennt, nachdem ein Fehler an die Methode `lostConnection` in `MqttCallback` zurückgegeben wurde, oder die Verbindung mit der Clientanwendung wird unterbrochen. Testen Sie, ob eine offene Verbindung vorhanden ist.
   c) Legen Sie über die Konstante `Example.quiesceTimeout` die maximale Zeit für die Stilllegung des Clients fest.

```
if (client.isConnected())
    client.disconnect(Example.quiesceTimeout);
```

Der Client wird beendet, wenn eine Kombination der folgenden drei Bedingungen eintritt:

a. Der Callback wurde für alle Nachrichten aufgerufen, die in dieser Sitzung veröffentlicht wurden bzw. in früheren Sitzungen, falls die Sitzung erneut gestartet wurde.
b. Nachrichten sind unvollständig und das Quiesce-Intervall ist abgelaufen. Das Quiesce-Intervall liegt standardmäßig bei 30 Sekunden. Sie können das Quiesce-Intervall ändern, indem Sie die Anzahl der zu wartenden Millisekunden als einen Parameter von `client.disconnect` übergeben.
c. `client.disconnect` wurde aufgerufen, nachdem einige Nachrichten veröffentlicht und vom Client in die Warteschlange gestellt wurden, aber bevor die Nachrichten gesendet wurden. Nachricht in der Warteschlange sind noch nicht unvollständig. Falls die Sitzung wieder anlauffähig ist, werden die Nachrichten nach dem Neustart der Sitzung erneut gesendet.

Ergebnisse

Wenn Sie die Veröffentlichungen unter Verwendung von WebSphere MQ anzeigen möchten, erstellen Sie ein Thema, eine Warteschlange und eine permanente Subskription, die alle als `"MQTTExampleTopic"` bezeichnet werden. Hierfür wird das Script in Abbildung 90 auf Seite 513 verwendet. Führen Sie den Client für die Veröffentlichung zum Thema `MQTT Examples` aus und führen Sie dann das Beispielprogramm **amqsbcg** aus, um die Veröffentlichungen in der `MQTTExamples` -Warteschlange zu durchsuchen.

1. Starten Sie einen Warteschlangenmanager und danach den zugehörigen Telemetrieservice (MQXR). Stellen Sie sicher, dass die TCP/IP-Adresse und der Port, die für den Telemetriekanal konfiguriert sind, den in der MQTT-Anwendung verwendeten Werten entsprechen.
2. Konfigurieren Sie eine permanente Subskription, indem Sie das Befehlsscript `mqttexamples.txt` erstellen und mit **runmqsc:** ausführen.

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Abbildung 90. `mqttExampleTopic.txt`

Zur Ausführung des Scripts unter Windows geben Sie folgenden Befehl ein:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Führen Sie den Client als Java-Anwendung in Eclipseoder durch Ausführen von Java in einem Befehlsfenster aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Anmerkung: Das Befehlsfenster muss in dem Verzeichnis geöffnet sein, in dem sich der Pfad befindet `com\ibm\mq\id`.

4. Zeigen Sie die Ergebnisse entweder mit WebSphere MQ Explorer oder durch Ausführung des folgenden Befehls an:

```
amqsbcg MQTTExampleQueue queue manager name
```

Beispielcode

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubAsync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      client.connect();
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 91. PubAsync.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 92. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Abbildung 93. Example.java

Wiederherstellbaren asynchronen Publisher für MQ Telemetry Transport mit Java erstellen

Bei dieser Task führen Sie ein Lernprogramm aus, um Ihre asynchrone Publisher-Anwendung zu ändern. Die Änderungen ermöglichen es der Anwendung, die Zustellung von Veröffentlichungen zu beenden, die bei der letzten Ausführung des Clients nicht bestätigt wurden.

Vorbereitende Schritte

Installieren Sie das Feature WebSphere MQ Telemetry auf einem Server, auf dem IBM WebSphere MQ Version 7.1 oder höher installiert ist.

Die Clientanwendung verwendet das `com.ibm.mq.micro.client.mqttv3`-Paket im IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Das SDK ist Bestandteil der IBM WebSphere MQ Telemetry-Installation. Der Client stellt eine Verbindung zu der IBM WebSphere MQ Telemetry-Funktion her, um Nachrichten mit IBM WebSphere MQ auszutauschen.

Sie müssen außerdem die Telemetrie-Aktualisierungen für IBM WebSphere MQ Explorer Version 7.1 installieren, um IBM WebSphere MQ Telemetry zu verwalten. Die Updates sind Bestandteil der IBM WebSphere MQ Telemetry-Installation.

Ein MQTT-Client, der in Java SE ausgeführt wird, erfordert Version 6.0 von Java SE oder höher. IBM Java SE v6.0 ist Teil der IBM WebSphere MQ Version 7.1 -Installation. Sie befindet sich unter *WebSphere MQ installation directory*\java\jre .

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Veröffentlichungsanwendung, PubAsyncRestartable. PubAsyncRestartable veröffentlicht Hello World zum Thema MQTT Examples, ohne auf die Bestätigung zu warten, dass die Veröffentlichung an den Warteschlangenmanager zugestellt wurde. Die Empfangsbestätigungen werden in einer Callback-Klasse (CallBack) empfangen. Alle Zustellungstokens für Veröffentlichungen, die in einer früheren Instanz nicht abgeschlossen wurden, können geprüft werden. Sie werden ebenfalls von der Callback-Klasse verarbeitet.

Wenn Sie eine permanente Subskription für MQTT Examples einrichten, können Sie überprüfen, ob die Anwendung funktioniert.

Die Prozedur verwendet Eclipse für die Entwicklung, Erstellung und Ausführung des Clients. Sie können Eclipse von der Website des Eclipse-Projekts unter www.eclipse.org herunterladen.

Die Schritte im Abschnitt Prozedur ändern die Anwendung PubAsync.java in „Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen” auf Seite 511.

Alternativ können Sie den Code unter „Beispielcode” auf Seite 520 in ein neues Verzeichnis (.\com\ibm\mq\id) kopieren. Erstellen Sie drei Java-Dateien: Example.java, CallBack.javaund PubAsyncRestartable.java. Kompilieren Sie die Beispiele mit folgendem Befehl:

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsyncRestartable.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Exam
ple.java
```

Führen Sie PubAsyncRestartable mithilfe des folgenden Befehls aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
     com.ibm.mq.id.PubAsyncRestartable.class
```

Vorgehensweise

1. Klicken Sie im Paket com.ibm.mq.id mit der rechten Maustaste auf die Datei PubAsync.java und kopieren Sie sie. Fügen Sie die Datei in dasselbe Paket ein und benennen Sie sie dabei in PubAsyncRestartable um.
2. Erstellen Sie eine wiederverwendbare Client-ID.

   ```
   Example.clientId = String.format(
       "%-23.23s",
       (System.getProperty("user.name") + "_" + (System.getProperty(
           "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
   ```

   Abbildung 94. Wiederverwendbare Client-ID

   Die Anwendungen in „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 und „Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen” auf Seite 511 verwendeten für jede Clientverbindung eine neue Client-ID. Für eine wieder anlauffähige Veröffentlichungs- oder Subskriptionsanwendung müssen Sie für jede einzelne Verbindung des Clients dieselbe Client-ID verwenden, für unterschiedliche Clients müssen jedoch unterschiedliche IDs verwendet werden (siehe „Client-ID” auf Seite 561). Die wiederverwendbare Client-ID wird aus dem Benutzernamen und dem Namen der Klasse gebildet. Ihre Länge ist auf 23 Bytes begrenzt. Sie darf nur Zeichen enthalten, die in Objektnamen für den Warteschlangenmanager gültig sind. Der Code entfernt alle Silbentrennungsstriche, die möglicherweise eingefügt wurden.

3. Die Servicequalität QoS der Nachricht wird auf 2 statt auf den Standartwert 1 gesetzt, um doppelte Nachrichten zu vermeiden.

```
message.setQos(Example.QoS);
```

   Sie müssen entweder den Wert von `Example.QoS` in 2ändern oder die Eigenschaft QoS als Argument mit der Option `-DQoS=2` in der Java-Befehlszeile übergeben.
4. Erstellen Sie ein `MqttConnectOptions`-Objekt und legen Sie das zugehörige Attribut `cleanSession` auf den Wert `false`.
   a) Erstellen Sie ein `MqttConnectOptions`-Objekt.

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

      `conOptions` ist ein Optionsparameter im Konstruktor `MqttClient`.
   b) Legen Sie das Attribut 'clearSession' fest.

```
conOptions.setCleanSession(Example.cleanSession);
```

      Der Parameter `Example.cleanSession` ist standardmäßig auf `true` gesetzt und entspricht damit der Standardeinstellung von `MqttConnectionOptions.cleanSession`.

   Wenn PubAsyncRestartable erneut gestartet wird, kann es mit einer "bereinigten Sitzung" starten und alle anstehenden Zustellungstokens für Nachrichten mit QoS 1 oder 2 löschen.

   Setzen Sie `Example.cleanSession` auf `false`, damit alle anstehenden Zustellungstokens beibehalten werden. Die Tokens werden von der Klasse `MqttCallBack` verarbeitet, wenn der Client erneut verbunden wird.
5. Rufen Sie nach dem erneuten Start der Sitzung alle anstehenden Zustellungstokens ab und drucken Sie deren Inhalte aus.

```
if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance "
            + client.getClientId());
```

6. Übergeben Sie den Parameter `conOptions` an den Konstruktor `MqttClient`.

```
client.connect(conOptions);
```

7. Legen Sie für Verbindungstrennungen ein maximales Trennungsintervall fest.

```
client.disconnect(Example.timeout);
```

   Damit anstehende Zustellungstokens, die verarbeitet werden, angezeigt werden können, muss eine frühere Instanz beendet werden, ohne dass die Zustellung abgeschlossen ist. Wenn das Beispiel ausgeführt werden soll, ohne dass Veröffentlichungen vor Beendigung von PubAsyncRestartable bestätigt werden, dann setzen Sie `Example.timeout` auf 0.

Ergebnisse

Wenn Sie die Veröffentlichungen unter Verwendung von WebSphere MQ anzeigen möchten, erstellen Sie ein Thema, eine Warteschlange und eine permanente Subskription, die alle als `"MQTTExampleTopic"` bezeichnet werden. Hierfür wird das Script in Abbildung 95 auf Seite 519 verwendet. Führen Sie den Cli-

ent für die Veröffentlichung zum Thema MQTT Examples aus und führen Sie dann das Beispielprogramm **amqsbcg** aus, um die Veröffentlichungen in der MQTTExamples -Warteschlange zu durchsuchen.

1. Starten Sie einen Warteschlangenmanager und danach den zugehörigen Telemetrieservice (MQXR). Stellen Sie sicher, dass die TCP/IP-Adresse und der Port, die für den Telemetriekanal konfiguriert sind, den in der MQTT-Anwendung verwendeten Werten entsprechen.
2. Konfigurieren Sie eine permanente Subskription, indem Sie das Befehlsscript `mqttexamples.txt` erstellen und mit **runmqsc:** ausführen.

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Abbildung 95. mqttExampleTopic.txt

Zur Ausführung des Scripts unter Windows geben Sie folgenden Befehl ein:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Führen Sie den Client als Java-Anwendung in Eclipseoder durch Ausführen von Java in einem Befehlsfenster aus:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Anmerkung: Das Befehlsfenster muss in dem Verzeichnis geöffnet sein, in dem sich der Pfad befindet `com\ibm\mq\id`.

4. Zeigen Sie die Ergebnisse entweder mit WebSphere MQ Explorer oder durch Ausführung des folgenden Befehls an:

```
amqsbcg MQTTExampleQueue queue manager name
```

Beispielcode

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
import com.ibm.micro.client.mqttv3.MqttDeliveryToken;
import com.ibm.micro.client.mqttv3.MqttMessage;
import com.ibm.micro.client.mqttv3.MqttTopic;
public class PubAsyncRestartable {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + (System.getProperty(
            "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance \""
            + client.getClientId() + "\"");
      client.connect(conOptions);
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 96. PubAsyncRestartable.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 97. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Abbildung 98. Example.java

Subskribenten für MQ Telemetry Transport mit Java erstellen

Bei dieser Task führen Sie ein Lernprogramm zum Erstellen einer Subskribentenanwendung aus. Der Subskribent erstellt eine Subskription für ein Thema und empfängt Veröffentlichungen für die Subskription.

Vorbereitende Schritte

Installieren Sie das Feature WebSphere MQ Telemetry auf einem Server, auf dem IBM WebSphere MQ Version 7.1 oder höher installiert ist.

Die Clientanwendung verwendet das `com.ibm.mq.micro.client.mqttv3`-Paket im IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Das SDK ist Bestandteil der IBM WebSphere MQ Telemetry-Installation. Der Client stellt eine Verbindung zu der IBM WebSphere MQ Telemetry-Funktion her, um Nachrichten mit IBM WebSphere MQ auszutauschen.

Sie müssen außerdem die Telemetrie-Aktualisierungen für IBM WebSphere MQ Explorer Version 7.1 installieren, um IBM WebSphere MQ Telemetry zu verwalten. Die Updates sind Bestandteil der IBM WebSphere MQ Telemetry-Installation.

Ein MQTT-Client, der in Java SE ausgeführt wird, erfordert Version 6.0 von Java SE oder höher. IBM Java SE v6.0 ist Teil der IBM WebSphere MQ Version 7.1 -Installation. Sie befindet sich unter *WebSphere MQ installation directory*\java\jre .

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Subskribentenanwendung, `Subscribe`. Mit `Subscribe` wird das Subskriptionsthema `MQTT Examples` erstellt und 30 Sekunden lang auf Veröffentlichungen zu dieser Subskription gewartet.

Ein Subskribent kann eine Subskription erstellen und auf Veröffentlichungen warten. Er kann auch Veröffentlichungen empfangen, die an eine zuvor erstellte Subskription gesendet werden, sofern dieselbe Client-ID verwendet wird. Über das boolesche Attribut `MqttConnectionOptions.cleanSession` wird gesteuert, ob zuvor gesendete Veröffentlichungen empfangen werden oder nicht (siehe „Subskriptionen" auf Seite 574).

Sie können die Veröffentlichungsprogramme verwenden, um Veröffentlichungen zu erstellen, oder Sie können WebSphere MQ Explorer verwenden, um eine Testveröffentlichung zum Thema `MQTT Examples` zu erstellen.

Die Prozedur verwendet Eclipse für die Entwicklung, Erstellung und Ausführung des Clients. Sie können Eclipse von der Website des Eclipse-Projekts unter www.eclipse.org herunterladen.

Bei den Anweisungen im Abschnitt Vorgehensweise wird davon ausgegangen, dass Sie das Paket `com.ibm.mq.id` bereits in einer der früheren Tasks erstellt und in die Klassen `Example.java` und `Callback.java` kopiert haben.

Vorgehensweise

1. Erstellen Sie die Klasse `Subscribe` im Paket `com.ibm.mq.id`.
2. Erstellen Sie eine wiederverwendbare Client-ID.

```
Example.clientId = String.format(
    "%-23.23s",
    (System.getProperty("user.name") + "_" + (System.getProperty(
        "clientId", "Subscribe."))).trim()).replace('-', '_');
```

Abbildung 99. Wiederverwendbare Client-ID

Die Anwendungen in „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen" auf Seite 505 und „Asynchronen Publisher für MQ Telemetry Transport mit Java erstellen" auf Seite 511 verwendeten für jede Clientverbindung eine neue Client-ID. Für eine wieder anlauffähige Veröffentlichungs- oder Subskriptionsanwendung müssen Sie für jede einzelne Verbindung des Clients dieselbe Client-ID verwenden, für unterschiedliche Clients müssen jedoch unterschiedliche IDs verwendet werden (siehe „Client-ID" auf Seite 561). Die wiederverwendbare Client-ID wird aus dem Benutzernamen und dem Namen der Klasse gebildet. Ihre Länge ist auf 23 Bytes begrenzt. Sie darf nur Zeichen enthalten, die in Objektnamen für den Warteschlangenmanager gültig sind. Der Code entfernt alle Silbentrennungsstriche, die möglicherweise eingefügt wurden.

3. Erstellen Sie einen `try-catch`-Block.

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

Der MQTT-Client gibt die Ausnahme `MqttException`, `MqttPersistenceException` oder `MqttSecurityException` aus. `MqttPersistenceException` und `MqttSecurityException` sind Unterklassen von `MqttException`.

Verwenden Sie die Methode `MqttException.getReasonCode`, um die Ursache der Ausnahme zu ermitteln. Verwenden Sie bei Ausgabe von `MqttPersistenceException` oder `MqttSecurityException` die Methode `getCause`, um die zugrundeliegende Throwable-Ausnahme zurückzugeben.

4. Erstellen Sie eine neue `MqttClient`-Instanz.

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

   Stellen Sie dem Client eine Serveradresse zur Verfügung, die später für die Verbindung mit WebSphere MQ verwendet werden soll. Legen Sie die Client-ID zur Benennung des Clients fest.

   - Optional können Sie eine Implementierung der Schnittstelle `MqttClientPersistence` bereitstellen, durch die die Standardschnittstelle ersetzt wird. Die Standardimplementierung `MqttPersistence` speichert QoS 1 und 2 Nachrichten, die auf die Zustellung warten, als Dateien; siehe „Nachrichtenpersistenz in MQTT-Clients" auf Seite 566.
   - Der standardmäßige IBM WebSphere MQ-TCP/IP-Port für MQTT lautet 1883. Für SSL ist es Port 8883. Im Beispiel wird die Standardadresse auf `tcp://localhost:1883` gesetzt.
   - Normalerweise muss ein bestimmter physischer Client anhand seiner Client-ID eindeutig zu erkennen sein. Die Client-ID muss daher für alle Clients, die eine Verbindung zu einem Server herstellen, eindeutig sein (siehe „Client-ID" auf Seite 561). Wenn für eine Instanz die gleiche Client-ID wie für eine frühere Instanz verwendet wird, bedeutet dies, dass die aktuelle Instanz eine Instanz des selben Clients ist. Wird eine Client-ID auf zwei aktiven Clients doppelt vergeben, wird auf beiden Clients eine Ausnahme ausgelöst und ein Client wird beendet.
   - Die Länge der Client-ID ist auf 23 Bytes beschränkt. Bei Überschreiten dieser Länge wird eine Ausnahme ausgegeben. Außerdem darf die Client-ID nur Zeichen enthalten, die auch in den Namen von Warteschlangenmanagern erlaubt sind. Bindestriche und Leerzeichen sind zum Beispiel nicht zulässig.
   - Eine Nachrichtenverarbeitung erfolgt erst, wenn Sie die Methode `MqttClient.connect` aufgerufen haben.

   Verwenden Sie das Clientobjekt zum Veröffentlichen und Subskribieren von Themen und zum Wiederherstellen von Informationen zu Veröffentlichungen, die noch nicht zugestellt wurden.
5. Erstellen Sie direkt vor der Codezeile `client.connect();` eine Instanz der Klasse `CallBack` unter Angabe der Client-ID.

```
CallBack callback = new CallBack(Example.clientId);
client.setCallback(callback);
```

   - Die Klasse `CallBack` implementiert `MqttCallBack`. Es ist eine Callback-Instanz pro Client-ID erforderlich. In diesem Beispiel übergibt der Konstruktor die zu speichernde Client-ID als Instanzdaten. Mit ihrer Hilfe wird im Callback ermittelt, welche Instanz des Callbacks gestartet wurde.
   - Sie müssen drei Methoden in der Callback-Klasse implementieren:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Empfängt eine Veröffentlichung, die subskribiert wurde.

     `public void connectionLost(Throwable cause)`
     Wird aufgerufen, wenn die Verbindung verloren geht.

     `public void deliveryComplete(MqttDeliveryToken token))`
     Wird aufgerufen, wenn ein Zustellungstoken für eine Nachricht mit Servicequalität 1 oder 2, die veröffentlicht wurde, empfangen wird.
   - Der Callback wird durch `MqttClient.connect` aktiviert.
6. Erstellen Sie ein `MqttConnectOptions`-Objekt und legen Sie das zugehörige `cleanSession`-Attribut fest.
   a) Erstellen Sie ein `MqttConnectOptions`-Objekt.

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

      `conOptions` ist ein Optionsparameter im Konstruktor `MqttClient`.
   b) Legen Sie das Attribut 'clearSession' fest.

```
conOptions.setCleanSession(Example.cleanSession);
```

Der Parameter `Example.cleanSession` ist standardmäßig auf `true` gesetzt und entspricht damit der Standardeinstellung von `MqttConnectionOptions.cleanSession`.

Wenn Sie die Standardeinstellung `MqttConnectOptions` verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor die Clientverbindung hergestellt wird, werden bei der Herstellung der Clientverbindung alle alten Subskriptionen für den Client gelöscht. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` auf `false` setzen, bevor eine Verbindung hergestellt wird, werden alle Subskriptionen, die der Client erstellt, zu den Subskriptionen hinzugefügt, die bereits vor Herstellung der Verbindung für den Client existierten. 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 `cleanSession`-Modus vor Herstellung der Verbindung festlegen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie den Modus von `cleanSession=false` in `cleanSession=true` ändern, werden alle bisherigen Subskriptionen für den Client gelöscht und alle noch nicht empfangenen Veröffentlichungen verworfen.

7. Übergeben Sie den Parameter `conOptions` an den Konstruktor `MqttClient`.

```
client.connect(conOptions);
```

8. Erstellen Sie eine Subskription.

```
      client.subscribe(Example.topicString, Example.QoS);
```

Im Beispiel wird eine `MqttClient.subscribe`-Methode zum Übergeben eines Themenfilters mit einer QoS-Option verwendet. Die Methode `MqttClient.subscribe` besitzt vier Signaturen und Sie können sowohl Feldgruppen von Subskriptionsfiltern als auch einen einzelnen Filter übergeben.

Im Beispiel wird ein Themenfilter verwendet, der von den Veröffentlichungsbeispielen als Themenfilter verwendet wurde, sodass alle von diesen Beispielen erstellten Veröffentlichungen empfangen werden.

Bei jeder Ausführung des Beispiels `subscribe.java` wird eine Subskription erstellt. Solange Sie 'Example.topicString' nicht ändern, erstellt es immer wieder dieselbe Subskription. Wenn eine Subskription wiederholt erstellt wird, führt dies nicht zu zwei identischen Subskriptionen. Ein Client empfängt keine doppelten Kopien von Veröffentlichungen, die mit ein und derselben Subskription übereinstimmen.

Subskriptionen werden im Abschnitt „Subskriptionen” auf Seite 574 und Filter im Abschnitt „Themenzeichenfolgen und Themenfilter in MQTT-Clients” auf Seite 576 beschrieben.

9. Warten Sie, bis einige Veröffentlichungen eintreffen, und trennen Sie dann die Clientverbindung.

```
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
```

Veröffentlichungen werden von der Implementierung der Methode `MqttCallback.messageArrived` empfangen.

Die Anwendung 'Subscribe' hat keine Nachrichten veröffentlicht und wartet deshalb nicht auf irgendwelche Zustellungstokens. `client.disconnect` wird ohne Verzögerung ausgeführt.

Beispielcode

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
public class Subscribe {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + System.getProperty("clientId",
            "Subscribe."))).trim();
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      client.connect(conOptions);
      System.out.println("Subscribing to topic \"" + Example.topicString
          + "\" for client instance \"" + client.getClientId()
          + "\" using QoS " + Example.QoS + ". Clean session is "
          + Example.cleanSession);
      client.subscribe(Example.topicString, Example.QoS);
      System.out.println("Going to sleep for " + Example.sleepTimeout / 1000
          + " seconds");
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
      System.out.println("Finished");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 100. Subscribe.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Abbildung 101. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Abbildung 102. Example.java

Zugehörige Konzepte

MQTT-Publish/Subscribe-Anwendungen

MQTT-Java-Client über JAAS authentifizieren

In diesem Abschnitt finden Sie Informationen zum Authentifizieren eines Clients über JAAS. Ändern Sie das Musterprogramm `JAASLoginModule.java` und das Java-Beispielprogramm `PubSync.java`. Konfigurieren Sie einen Telemetriekanal, um die JAAS-Authentifizierung anzufordern, und führen Sie den geänderten Publisher aus, wobei Sie den zugehörigen `Benutzernamen` und das zugehörige `Kennwort` mithilfe von JAAS prüfen.

Vorbereitende Schritte

Es wird vorausgesetzt, dass Sie vor Ausführung dieser Übung die MQTT-V3-Client-JAR-Dateien, Javadoc und Eclipse installiert, Telemetriekanäle konfiguriert sowie PubSync.java codiert und ausgeführt haben. Sie verfügen über einen Eclipse-Arbeitsbereich, in dem sich eine aktiver Version von PubSync.java befindet.

Die Übung wurde für Windows geschrieben. Ändern Sie die Verzeichnispfade für Linux.

Informationen zu diesem Vorgang

Die Task basiert auf der Änderung der JAASLoginModule -Beispielklasse in *WMQ Installation directory*\mqxr\samples\JAASLoginModule.java , um MyLogin.javazu erstellen. In der Task ändern Sie außerdem den Beispielcode PubSync.java in „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505, um einen Benutzernamen und ein Kennwort festzulegen. Zu Testzwecken wird MyLogin.java den Benutzernamen und das Kennwort nach dem Zufallsprinzip akzeptieren oder ablehnen.

Die Schritte in dieser Task sind als Programmierungsübung geschrieben. Sie müssen die Prozedur anpassen, um eine echte Authentifizierung in einer Produktionsumgebung ausführen zu können.

In einer typischen Erläuterung zur Vorgehensweise bei der Programmierung der JAAS-Authentifizierung wird vorausgesetzt, dass das Anmeldemodul den Kontext, der JAAS geladen hat, authentifiziert. Wenn der Telemetrieservice (MQXR) JAAS aufruft, ist der Telemetrieservice der Kontext, der JAAS geladen hat. Es hat keinen Sinn, den Telemetrieservicekontext zu authentifizieren; es ist immer mqm. Stattdessen richtet der Telemetrieservice den Benutzernamen und das Kennwort für den Client ein, sodass sie für die Anmeldemodulklasse verfügbar sind. Der Benutzername und das Kennwort werden mithilfe von zwei Callbacks an das Anmeldemodul übergeben.

```
javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
callbacks[0] =
      new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] =
      new javax.security.auth.callback.PasswordCallback("PasswordCallback", false);
callbackHandler.handle(callbacks);
String username =
      ((javax.security.auth.callback.NameCallback) callbacks[0]).getName();
char[] password =
      ((javax.security.auth.callback.PasswordCallback) callbacks[1]).getPassword();
```

Der Benutzername und das Kennwort des Clients sind die einzigen Informationen über den Client, die für das Anmeldemodul verfügbar sind.

Vorgehensweise

1. Erstellen Sie zwei Pakete, samples und security.jaas, im selben Java-Projekt wie PubSync.java.

   Das Paket samples wird nur als Referenz verwendet. Führen Sie die Codeänderungen im Paket security.jaas durch.
2. Importieren Sie JAASLoginModule.java und JAASPrincipal.java in beide Pakete.

   Falls nötig, refaktorieren Sie die Paketierungsanweisungen in der Java-Quelle, um Kompilierungsfehler zu beseitigen.
3. Refaktorieren Sie den Klassennamen JAASLoginModule im Paket security.jaas zu MyLogin
4. Ersetzen Sie in MyLogin.java einen Teil des Codes in der Methode login, um anzuzeigen, dass das Modul aktiv ist.
   a) Ersetzen Sie den Code

   ```
   // Accept everything.
   if (true)
     loggedIn = true;
   else
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   ```

   b) durch folgenden Code:

   ```
   // login half the users randomly
   PrintWriter pw = new PrintWriter(new FileWriter(System.getProperty("user.dir")
           + "\\MyLogin.log", true));
   pw.println("Called JAASLogin.login at "
           + System.getProperty("publication", "Hello World "
           + String.format("%tc", System.currentTimeMillis())));
   if (Math.random() < 0.5)
   ```

```
  loggedIn = true;
pw.println("Username: \"" + username + "\", Password: \""
    + String.valueOf(password) + "\" loggedIn: " + loggedIn);
pw.close();
if (!loggedIn)
  throw new javax.security.auth.login.FailedLoginException("Login failed");
principal= new JAASPrincipal(username);
```

Die vollständige Quelle für `MyLogin.java` ist im Abschnitt Abbildung 105 auf Seite 531 enthalten. Die Quelle für `JAASPrincipal.java` mit dem Paketnamen, der in `security.jaas` refaktoriert wurde, ist im Abschnitt Abbildung 106 auf Seite 532 enthalten.

5. Legen Sie den Klassenpfad in `service.env` so fest, dass er auf das Verzeichnis verweist, das den Pfad zu `security/jaas/MyLogin.class` und `security/jaas/JAASPrincipal.class` enthält.

```
CLASSPATH=C:\WMQTelemetryApps\MQTTSecureExamples\bin
```

Im Abschnitt JAAS-Konfiguration für den Telemetriekanal finden Sie Informationen dazu, wie Sie mithilfe von `service.env` einen Klassenpfad an einen WebSphere MQ-Service übergeben können.

6. Fügen Sie eine Anmeldemodulzeilengruppe zur Datei `jaas.config` hinzu.

```
MyLoginExample {
  security.jaas.MyLogin required debug=true;
  };
```

Im Abschnitt JAAS-Konfiguration für den Telemetriekanal finden Sie Informationen darüber, wie ein JAAS-Anmeldemodul in `jaas.config` definiert wird.

7. Fügen Sie mithilfe des Assistenten **Neuer Telemetriekanal** in WebSphere MQ Explorer einen Telemetriekanal hinzu und konfigurieren Sie den Kanal so, dass er eine JAAS-Authentifizierung anfordert. Stellen Sie einen Bezug zur Zeilengruppe `MyLoginExample` her.

Passen Sie beispielsweise die Informationen an, die Sie aus der folgenden Zeilengruppe in der Datei `mqxr_win.properties` in den Assistenten eingeben. Wenn Sie unter Linux arbeiten, hat die Datei die Bezeichnung `mqxr_unix.properties`. Bearbeiten Sie die Telemetrieeigenschaftendatei nicht direkt, sondern verwenden Sie den Assistenten.

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=MyLoginExample;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Anmerkung: Falls Sie einen der Telemetriekanalparameter oder die Klasse `security.jaas.MyLogin` ändern, müssen Sie den Telemetrieservice stoppen und erneut starten. Die Änderungen werden nur wirksam, wenn Sie den Service erneut starten.

8. Erstellen Sie eine Kopie der Datei PubSync`.java` im Paket `com.ibm.mq.id` und geben Sie der Kopie die Bezeichnung PubSyncJAAS`.java`.

Im Abschnitt „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 wird die Vorgehensweise zum Erstellen von PubSync.java im Paket `com.ibm.mq.id` beschrieben.

9. Legen Sie den Wert für `MqttConnectOptions.username` und `MqttConnectOptions.password` im Programm PubSyncJAAS`.java` fest und übergeben Sie `MqttConnectOptions` als Parameter von `MqttClient.connect`.

```
MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setUserName(Example.username);
conOptions.setPassword(Example.password);
client.connect(conOptions);
```

Überprüfen Sie den kursiv dargestellten Code in PubSyncJAAS.Java, in dem die in Example.java festgelegten Konstanten verwendet werden.

10. Setzen Sie `Example.TCPAddress` auf die Socketadresse des Telemetriekanals, den Sie für die Verwendung für die JAAS-Konfiguration `MyLoginExample` konfiguriert haben. Verwenden Sie beispielsweise die Portnummer 1884.
11. Führen Sie `PubSyncJAAS` mehrere Male aus, um zu sehen, wie sich der Client anmeldet und ob die Anmeldung akzeptiert oder abgelehnt wird.

    Bei jeder Ablehnung des Anmeldeversuchs wird eine Ausnahmebedingung ausgelöst.

Ergebnisse

Abbildung 103 auf Seite 530 zeigt die Ergebnisse einer zweimaligen Ausführung von PubSyncJAAS.Java. Die Protokolleinträge werden in Abbildung 104 auf Seite 530 gezeigt.

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:05 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_61c57a18_4bf7_40d" on address tcp://local
host:1884"
With username "Admin" and password "Password"
Client exception caught
Client is not connected (32104)
    at com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHel
per.java:33)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:88)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:105)
    at com.ibm.micro.client.mqttv3.MqttTopic.publish(MqttTopic.java:68)
    at com.ibm.mq.id.PubSync.main(PubSync.java:24)
```

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:40 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_1d1599a0_50f5_4ea" on address tcp://local
host:1884"
With username "Admin" and password "Password"
Delivery token "1731749688" has been received: true
```

Abbildung 103. Konsolenausgabe von PubSyncJAAS.java

Die Protokolldatei `MyLogin.log` wird in *WMQ Data directory*gespeichert. Beispiel: `C:\IBM\MQ\Data\MyLogin.log`:

```
Called JAASLogin.login at Hello World Fri Jun 04 08:31:05 BST 2010
Username: "Admin", Password: "Password" loggedIn: false
Called JAASLogin.login at Hello World Fri Jun 04 08:31:40 BST 2010
Username: "Admin", Password: "Password" loggedIn: true
```

Abbildung 104. MyLogin.log

Beispiele

Der kursiv dargestellte Code in Abbildung 105 auf Seite 531 zeigt die Änderung im Beispiel `JAASLoginModule.java`.

```
package security.jaas;
import java.io.FileWriter;
import java.io.PrintWriter;

public class JAASLogin implements javax.security.auth.spi.LoginModule {
  private javax.security.auth.Subject                  subject;
  private javax.security.auth.callback.CallbackHandler callbackHandler;
  JAASPrincipal                                        principal;
  boolean                                              loggedIn = false;
  public void initialize(javax.security.auth.Subject subject,
      javax.security.auth.callback.CallbackHandler callbackHandler,
      java.util.Map<String, ?> sharedState, java.util.Map<String, ?> options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
  }
  public boolean login() throws javax.security.auth.login.LoginException {
    try {
      javax.security.auth.callback.Callback[] callbacks = new javax.security.auth.callback.Call
back[2];
      callbacks[0] = new javax.security.auth.callback.NameCallback(
          "NameCallback");
      callbacks[1] = new javax.security.auth.callback.PasswordCallback(
          "PasswordCallback", false);

      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // login half the users randomly
      PrintWriter pw = new PrintWriter(new FileWriter(System
          .getProperty("user.dir")
          + "\\mylogin.log", true));
      pw.println("Called JAASLogin.login at "
          + System.getProperty("publication", "Hello World "
              + String.format("%tc", System.currentTimeMillis())));
      if (Math.random() < 0.5)
        loggedIn = true;
      pw.println("Username: \"" + username + "\", Password: \""
          + String.valueOf(password) + "\" loggedIn: " + loggedIn);
      pw.close();
      if (!loggedIn)
        throw new javax.security.auth.login.FailedLoginException("Login failed");
      principal = new JAASPrincipal(username);
    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }
    return loggedIn;
  }
  public boolean abort() throws javax.security.auth.login.LoginException {
    logout();
    return true;
  }
  public boolean commit() throws javax.security.auth.login.LoginException {
    if (loggedIn) {
      if (!subject.getPrincipals().contains(principal))
        subject.getPrincipals().add(principal);
    }
    return true;
  }
  public boolean logout() throws javax.security.auth.login.LoginException {
    subject.getPrincipals().remove(principal);
    principal = null;
    loggedIn = false;
    return true;
  }
}
```

Abbildung 105. MyLogin.java

Abbildung 106 auf Seite 532 ist der Beispielcode JAASLoginPrincipal.java, der in das Paket security.jaas kopiert wurde. Der Zweck von JAASLoginPrincipal ist die Implementierung der

`java.security.Principal`-Schnittstelle, um eine Liste der Benutzer aufzuzeichnen, die erfolgreich von `MyLogin` angemeldet wurden.

```
package security.jaas;
public class JAASPrincipal implements java.security.Principal,
    java.io.Serializable {
  private static final long serialVersionUID = 1L;
  String                    name;
  public JAASPrincipal(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return (name);
  }
  public boolean equals(Object object) {
    if (object != null && object instanceof JAASPrincipal
        && name.equals(((JAASPrincipal) object).getName()))
      return true;
    else
      return false;
  }
  public int hashCode() {
    return name.hashCode();
  }
}
```

Abbildung 106. JAASLoginPrincipal.java

Der Code in PubSync.java, der geändert wurde, um einen `Benutzernamen` und ein `Kennwort` hinzuzufügen, wird in Abbildung 107 auf Seite 532 kursiv dargestellt.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setUserName(Example.username);
      conOptions.setPassword(Example.password);
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("With username \"" + conOptions.getUserName()
              + "\" and password \"" + String.valueOf(conOptions.getPassword()) + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Abbildung 107. PubSyncJAAS.Java

Ändern Sie die Konstanten in Example.java, um Sie an Ihre Konfiguration anzupassen. Ignorieren Sie für dieses Beispiel die SSL-Einstellungen.

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Abbildung 108. Example.java

SSL-Telemetrieverbindung über selbst signierte Zertifikate authentifizieren

Authentifizieren Sie eine SSL-Verbindung mit selbst signierten Zertifikaten, mit mithilfe von **Keytool** generiert wurden. Sie haben die Möglichkeit, nur den Telemetriekanal oder den Telemetriekanal und die damit verbundenen Clients zu authentifizieren. Der Nachrichtenfluss auf der Verbindung wird verschlüsselt.

Vorbereitende Schritte

Führen Sie vor dem Start die Task „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 aus, damit PubSync.java mit einer nicht gesicherten TCP/IP-Verbindung ausgeführt wird. In dieser Task ändern Sie PubSync.java so, dass eine SSL-Verbindung verwendet wird.

Informationen zu diesem Vorgang

Die Schritte in dieser Task sind als Programmierungsübung geschrieben. Sie müssen die Prozedur anpassen, um eine echte Authentifizierung in einer Produktionsumgebung ausführen zu können.

Die Übung wurde für Windows geschrieben. Ändern Sie die Verzeichnispfade für Linux.

Vorgehensweise

1. Führen Sie die Task „'PubSync.java' für die Verwendung von SSL ändern” auf Seite 534 aus, um PubSync.java für die Verwendung von SSL zu ändern.
2. Konfigurieren Sie den Telemetriekanal und erstellen Sie die Schlüsselspeicher für die Verwendung von SSL.

   Authentifizieren Sie entweder nur den Telemetriekanal oder den Kanal und die damit verbundenen Clients:

   - Führen Sie die Task „Telemetriekanal authentifizieren” auf Seite 535 aus, um eine SSL-Verbindung herzustellen und den Telemetriekanal zu authentifizieren.
   - Führen Sie die Task „Telemetriekanal und Clients authentifizieren” auf Seite 536 aus, um eine SSL-Verbindung herzustellen und den Telemetriekanal und die damit verbundenen Clients zu authentifizieren.
3. Stoppen Sie den Telemetrieservice (MQXR) und starten Sie ihn erneut, damit die Änderungen in den Telemetriekanalkonfigurationen wirksam werden.
4. Führen Sie das Clientprogramm aus, um festzustellen, ob die Konfiguration funktioniert.

'PubSync.java' für die Verwendung von SSL ändern

Ändern Sie das erste Publisher-Programmbeispiel, um eine Verbindung zu einem Telemetriekanal über SSL herzustellen. Legen Sie die SSL-Eigenschaften fest, die vom geänderten Programm verwendet werden.

Vorbereitende Schritte

Es wird vorausgesetzt, dass Sie vor Ausführung dieser Übung die MQTT-V3-Client-JAR-Dateien, Javadoc und Eclipse installiert, Telemetriekanäle konfiguriert sowie PubSync.java codiert und ausgeführt haben. Sie verfügen über einen Eclipse-Arbeitsbereich, in dem sich eine aktiver Version von PubSync.java befindet.

Informationen zu diesem Vorgang

In der Task wird der Publisher-Client PubSync.java verwendet, den Sie im Abschnitt „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 als Basis erstellt haben. Für die Verwendung von SSL sind nur wenige Änderungen erforderlich (siehe Abbildung 109 auf Seite 535 und Abbildung 110 auf Seite 535).

Vorgehensweise

1. Erstellen Sie eine Kopie der Datei `PubSync.java` im Paket `com.ibm.mq.id` und geben Sie der Kopie die Bezeichnung `PubSyncSSL.java`.

   Im Abschnitt „Erste MQ Telemetry Transport-Publisher-Anwendung mit Java erstellen” auf Seite 505 wird die Vorgehensweise zum Erstellen von PubSync.java im Paket `com.ibm.mq.id` beschrieben.
2. Setzen Sie `Example.SSLAddress` auf die Socketadresse des Telemetriekanals, den Sie für die Verwendung für die SSL-Konfiguration konfiguriert haben.
3. Ändern Sie den Socketadressenparameter des Clientkonstruktors, sodass `Example.SSLAddress` verwendet wird.

   ```
   MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
   ```
4. Legen Sie den Wert für `MqttConnectOptions.SSLProperties` in `PubSyncSSL.java` fest und übergeben Sie `MqttConnectOptions` als Parameter von `MqttClient.connect`.

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setSSLProperties(Example.getSSLSettings());
   client.connect(conOptions);
   ```

Überprüfen Sie den kursiv dargestellten Code in PubSyncSSL.Java, in dem die in Example.java festgelegten Konstanten verwendet werden.

Beispiele

Die Änderungen an PubSync.java zum Hinzufügen von SSL werden unter Abbildung 109 auf Seite 535 in Kursivschrift dargestellt.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setSSLProperties(Example.getSSLSettings());
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("SSL Properties" + conOptions.getSSLProperties());
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Abbildung 109. PubSyncSSL.Java

Die Änderungen an Example.java werden in Abbildung 110 auf Seite 535 gezeigt.

```
 public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");

 public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore", "C:\\Certificates\\SSClientKey.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword", "password");
    properties.setProperty("com.ibm.ssl.trustStore", "C:\\Certificates\\SSClientTrust.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword", "password");
    return properties;
```

Abbildung 110. Änderungen an Example.java

Telemetriekanal authentifizieren

Clients authentifizieren den Telemetriekanal, um den Inhalt der Nachrichten, die durch den Kanal fließen, zu verschlüsseln und um sicherzustellen, dass ein Client eine Verbindung mit dem richtigen Telemetriekanal herstellt. Der Server führt keine Authentifizierung des Clients durch.

Informationen zu diesem Vorgang

Zum Erstellen und Verwalten von selbst signierten Zertifikaten können Sie verschiedene Schlüsselspeichereditoren verwenden. In der Task wird der Befehlszeilenbefehl **keytool** verwendet, der Teil der JRE ist. Sie können das mit WebSphere MQ gelieferte grafische Benutzerschnittstellentool **iKeyman** verwen-

den, um Schlüsselspeicher und generierte Schlüssel anzuzeigen. Starten Sie **iKeyman** mit dem Befehl **strmqikm**.

Vorgehensweise

1. Erstellen Sie mit dem Assistenten **New telemetry channel** (Neuer Telemetriekanal) einen Telemetriekanal mit der Bezeichnung SSLSSOptClients, für den eine SSL-Verbindung erforderlich ist. Der Kanal akzeptiert anonyme Clients.

   Passen Sie Ihre Kanalkonfiguration auf Basis der folgenden Konfigurationszeilengruppe an. Bearbeiten Sie die Telemetrieeigenschaftendatei nicht direkt, sondern verwenden Sie den Assistenten.

   ```
   com.ibm.mq.MQXR.channel/SSLSSOptClients: \
   com.ibm.mq.MQXR.Port=8883;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Generieren Sie die Schlüssel für den Client zur Authentifizierung des Telemetriekanals.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für den Telemetriekanal in einem neuen Schlüsselspeicher mit dem Namen SSServerOptKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerOptKey.jks -storepass password -keypass password
   ```

   b) Exportieren Sie das zugehörige öffentliche Zertifikat als ASCII-Datei mit der Option -rfc:

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerOptKey.jks -storepass password -rfc
   ```

   Wenn Sie die Task unter Windows ausführen, klicken Sie doppelt auf SSServerPublic.cer, um die Inhalte zu prüfen.

   c) Importieren Sie das öffentliche Zertifikat in einen neuen Client-Truststore, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

   d) Erstellen Sie einen leeren Client-Schlüsselspeicher, SSClientKey.jks.

   Keytool verfügt über keinen Befehl zum Erstellen eines leeren Schlüsselspeichers. Sie haben zwei Möglichkeiten:

   i) Führen Sie **strmqikm** aus und erstellen Sie einen Schlüsselspeicher (SSClientKey.jks), fügen aber keine Schlüssel hinzu.

   ii) Führen Sie Schritt 3a in „Telemetriekanal und Clients authentifizieren” auf Seite 536 aus, verwenden Sie die Schlüssel aber noch nicht.

Telemetriekanal und Clients authentifizieren

Clients authentifizieren den Telemetriekanal und der Telemetriekanal authentifiziert die mit ihm verbundenen Clients. Der Nachrichtenfluss im Kanal wird verschlüsselt.

Informationen zu diesem Vorgang

Zum Erstellen und Verwalten von selbst signierten Zertifikaten können Sie verschiedene Schlüsselspeichereditoren verwenden. In der Task wird der Befehlszeilenbefehl **keytool** verwendet, der Teil der JRE ist. Sie können das mit WebSphere MQ gelieferte grafische Benutzerschnittstellentool **iKeyman** verwenden, um Schlüsselspeicher und generierte Schlüssel anzuzeigen. Starten Sie **iKeyman** mit dem Befehl **strmqikm**.

Der Telemetriekanal wird mit einem anderen Schlüsselspeicher als in der Task „Telemetriekanal authentifizieren” auf Seite 535 konfiguriert. Sie können denselben Schlüsselspeicher verwenden und den Schritt „2” auf Seite 537 zum Hinzufügen von Schlüsseln zum Schlüsselspeicher überspringen.

Vorgehensweise

1. Erstellen Sie mit dem Assistenten **New telemetry channel** (Neuer Telemetriekanal) einen Telemetriekanal mit der Bezeichnung SSLSSReqClients, für den eine SSL-Verbindung erforderlich ist. Der Kanal akzeptiert nur authentifizierte Clients.

   Passen Sie Ihre Kanalkonfiguration auf Basis der folgenden Konfigurationszeilengruppe an:

   ```
   com.ibm.mq.MQXR.channel/SSLSSReqClients: \
   com.ibm.mq.MQXR.Port=8884;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Generieren Sie die Schlüssel für den Client zur Authentifizierung des Telemetriekanals.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für den Telemetriekanal in einem neuen Schlüsselspeicher mit dem Namen SSServerReqKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerReqKey.jks -storepass password -keypass password
   ```

   b) Exportieren Sie das zugehörige öffentliche Zertifikat als ASCII-Datei mit der Option -rfc:

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerReqKey.jks -storepass password -rfc
   ```

   Wenn Sie die Task unter Windows ausführen, klicken Sie doppelt auf SSServerPublic.cer, um die Inhalte zu prüfen.

   c) Importieren Sie das öffentliche Zertifikat in einen neuen Client-Truststore, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

3. Generieren Sie die Schlüssel für den Telemetriekanal zum Authentifizieren eines Clients.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für den Client in einem neuen Schlüsselspeicher, SSClientKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSClientPrivate
           -dname "CN=mqttclient.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSClientKey.jks -storepass password -keypass password
   ```

   b) Exportieren Sie das zugehörige öffentliche Zertifikat als ASCII-Datei mit der Option -rfc:

   ```
   Keytool -export -noprompt -alias SSClientPrivate -file SSClientPublic.cer
           -keystore SSClientKey.jks -storepass password -rfc
   ```

   Wenn Sie die Task unter Windows ausführen, klicken Sie doppelt auf SSClientPublic.cer, um die Inhalte zu prüfen.

   c) Importieren Sie das öffentliche Zertifikat in den neuen Server-Schlüsselspeicher, SSServerReqKey.jks:

   ```
   Keytool -import -noprompt -alias SSClientPublic -file SSClientPublic.cer
           -keystore SSServerReqKey.jks -storepass password
   ```

   Telemetriekanäle verwenden für private Schlüssel und vertrauenswürdige Zertifikate den gleichen Speicher.

SSL-Telemetrieverbindung über eine Zertifikatskette authentifizieren

Verwenden Sie signierte Zertifikate, die Sie von einer Zertifizierungsstelle oder durch Implementierung eines eigenen Zertifizierungsverfahrens erhalten haben, zur Authentifizierung einer SSL-Verbindung. Sie haben die Möglichkeit, nur den Telemetriekanal oder den Telemetriekanal und die damit verbundenen Clients zu authentifizieren. Der Nachrichtenfluss auf der Verbindung wird verschlüsselt.

Vorbereitende Schritte

Führen Sie, bevor Sie beginnen, die Übung „SSL-Telemetrieverbindung über selbst signierte Zertifikate authentifizieren" auf Seite 533 aus, sodass `PubSyncSSL.Java` mit einer gesicherten TCP/IP-Verbindung unter Verwendung selbst signierter Zertifikate arbeitet.

Informationen zu diesem Vorgang

Ändern Sie in dieser Task die Tasks „Telemetriekanal authentifizieren" auf Seite 535 und „Telemetriekanal und Clients authentifizieren" auf Seite 536 im Abschnitt „SSL-Telemetrieverbindung über selbst signierte Zertifikate authentifizieren" auf Seite 533, um mit Schlüsseln zu arbeiten, die von einer Zertifikatskette zertifiziert wurden.

Sie können die Zertifikate für diese Task von einer Zertifizierungsstelle oder von einer Website wie beispielsweise http://www.openca.org/ abrufen. Kommerzielle Zertifizierungsstellen stellen im Allgemeinen kostenlose Testzertifikate für einen kurzen Zeitraum aus. Diese Task wurde mit kommerziell erlangten Zertifikaten getestet.

Eine andere Möglichkeit besteht darin, ein eigenes Zertifizierungsverfahren einzurichten und auf Ihren Computern auszuführen. Verwenden Sie dazu Tools von Websites wie beispielsweise https://www.openssl.org/.

Die `cacerts`-Truststores für JRE werden in dieser Task nicht verwenden. Sie können den `cacerts`-Truststore für JRE für den Client in der Task (siehe „Telemetriekanal authentifizieren" auf Seite 538) anstelle des angegebenen Truststores verwenden. Die Zertifikatskette kann von einer bekannten Zertifizierungsstelle signiert werden, deren Stammzertifikat sich bereits im `cacerts`-Speicher des Clients befindet. In diesem Fall geben Sie im Client keinen Truststore an. Wenn mehrere JREs im Client installiert sind, stellen Sie sicher, dass der korrekte `cacerts`-Speicher verwaltet wird.

Vorgehensweise

1. Falls noch nicht geschehen, führen Sie die Task „'PubSync.java' für die Verwendung von SSL ändern" auf Seite 534 durch, um PubSync.java für die Verwendung von SSL zu ändern.
2. Konfigurieren Sie den Telemetriekanal und erstellen Sie die Schlüsselspeicher für die Verwendung von SSL.

   Authentifizieren Sie entweder nur den Telemetriekanal oder den Kanal und die damit verbundenen Clients:

   - Führen Sie die Task „Telemetriekanal authentifizieren" auf Seite 538 aus, um eine SSL-Verbindung herzustellen und den Telemetriekanal zu authentifizieren.
   - Führen Sie die Task „Telemetriekanal und Clients authentifizieren" auf Seite 540 aus, um eine SSL-Verbindung herzustellen und den Telemetriekanal und die damit verbundenen Clients zu authentifizieren.
3. Stoppen Sie den Telemetrieservice (MQXR) und starten Sie ihn erneut, damit die Änderungen in den Telemetriekanalkonfigurationen wirksam werden.
4. Führen Sie das Clientprogramm aus, um festzustellen, ob die Konfiguration funktioniert.

Telemetriekanal authentifizieren

Clients authentifizieren den Telemetriekanal, um den Inhalt der Nachrichten, die durch den Kanal fließen, zu verschlüsseln und um sicherzustellen, dass ein Client eine Verbindung mit dem richtigen Telemetriekanal herstellt. Der Server führt keine Authentifizierung des Clients durch.

Informationen zu diesem Vorgang

Zum Erstellen und Verwalten von Zertifikaten können Sie verschiedene Schlüsselspeichereditoren verwenden. In der Task wird der Befehlszeilenbefehl **keytool** verwendet, der Teil der JRE ist. Sie können das mit WebSphere MQ gelieferte grafische Benutzerschnittstellentool **iKeyman** verwenden, um Schlüsselspeicher und generierte Schlüssel anzuzeigen. Starten Sie **iKeyman** mit dem Befehl **strmqikm**.

Vorgehensweise

1. Erstellen Sie mit dem Assistenten **New telemetry channel** (Neuer Telemetriekanal) einen Telemetriekanal mit der Bezeichnung SSLCAOptClients, für den eine SSL-Verbindung erforderlich ist. Der Kanal akzeptiert anonyme Clients.

   Passen Sie Ihre Kanalkonfiguration auf Basis der folgenden Konfigurationszeilengruppe an. Bearbeiten Sie die Telemetrieeigenschaftendatei nicht direkt, sondern verwenden Sie den Assistenten.

   ```
   com.ibm.mq.MQXR.channel/SSLCAOptClients: \
   com.ibm.mq.MQXR.Port=8885;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Generieren Sie einen von einer Zertifizierungsstelle signierten Schlüssel für den Client zur Authentifizierung des Telemetriekanals.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für den Telemetriekanal in einem neuen Schlüsselspeicher mit dem Namen SSServerOptKey.jks:

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverOpt.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   Der Schlüsselalgorithmus ist auf RSA gesetzt, weil dies von einigen Zertifizierungsstellen gefordert wird. Der allgemeine Name des Zertifikats muss eindeutig sein, denn einige Zertifizierungsstellen stellen keine Schlüssel mit gleichlautenden allgemeinen Namen aus.

   b) Erstellen Sie eine Zertifikatssignieranforderung in Form einer ASCII-Datei

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerOptKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   c) Führen Sie die Software der Zertifizierungsstelle aus oder melden Sie sich bei deren Website an. Fügen Sie die Inhalte von CAServerOptKey.csr ein, wenn Sie nach der Datei mit der Zertifikatssignieranforderung gefragt werden.

   d) Die Zertifizierungsstelle gibt ein oder zwei Zertifikate und eine signierte Antwortdatei als ASCII-Dateien zurück. Fügen Sie den Inhalt in zwei oder drei Dateien ein:

   Stammzertifikat
   Fügen Sie dies in CARoot.cer ein.

   Zwischenzertifikat
   Fügen Sie dies in CAInter.cer ein.

   Server-signierte Antwortdatei
   Fügen Sie dies in CAServerOpt.rsp ein.

   Der JRE-Zertifikatsspeicher wird in dieser Task nicht verwendet. Wenn Sie ein einziges Stammzertifikat und eine signierte Antwort von der Zertifizierungsstelle empfangen haben, verwenden Sie das Stammzertifikat und die signierte Antwort in den folgenden Schritten. Wenn Sie ein Stamm- und ein Zwischenzertifikat empfangen haben, verwenden Sie das Zwischenzertifikat und die signierte Antwort.

e) Empfangen Sie die signierte Serverantwort in dem Serverschlüsselspeicher, aus dem Sie die Zertifikatsanforderung gestellt haben.

Durch das Empfangen der Antwort wird das selbst signierte Zertifikat in ein von der Zertifizierungsstelle signiertes Zertifikat geändert. Wenn Sie sich das Zertifikat vor und nach dem Empfang der Antwort im Schlüsselspeicher anschauen, werden Sie feststellen, dass der Unterzeichner entsprechend geändert wird. Falls nicht, meldet das Schlüsselverwaltungstool einen Fehler. Überprüfen Sie das Zertifikat, bevor Sie es verwenden, und vergewissern Sie sich, dass jetzt die Zertifizierungsstelle als Unterzeichner ausgewiesen wird.

```
Keytool -import -noprompt -alias CAServer -file CAServerOpt.rsp
        -keystore CAServerOptKey.jks -storepass password
```

In einigen Schlüsselverwaltungsprogrammen, z. B. in **iKeyman** empfangen Sie Antwortdateien, statt sie zu importieren.

f) Importieren Sie das Zertifikat der Zertifizierungsstelle in den Client-Truststore.

Importieren Sie entweder das Zwischenzertifikat, falls Sie zwei Zertifikate von der Zertifizierungsstelle empfangen haben, oder das Stammzertifikat, falls Sie nur ein Zertifikat empfangen haben.

Entweder:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

Oder:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

Telemetriekanal und Clients authentifizieren

Clients authentifizieren den Telemetriekanal und der Telemetriekanal authentifiziert die mit ihm verbundenen Clients. Der Nachrichtenfluss im Kanal wird verschlüsselt.

Informationen zu diesem Vorgang

Zum Erstellen und Verwalten von Zertifikaten können Sie verschiedene Schlüsselspeichereditoren verwenden. In der Task wird der Befehlszeilenbefehl **keytool** verwendet, der Teil der JRE ist. Sie können das mit WebSphere MQ gelieferte grafische Benutzerschnittstellentool **iKeyman** verwenden, um Schlüsselspeicher und generierte Schlüssel anzuzeigen. Starten Sie **iKeyman** mit dem Befehl **strmqikm**.

Der Telemetriekanal wird mit einem anderen Schlüsselspeicher als in der Task „Telemetriekanal authentifizieren” auf Seite 538 konfiguriert. Sie können denselben Schlüsselspeicher verwenden und den Schritt „2” auf Seite 541 zum Hinzufügen von Schlüsseln zum Schlüsselspeicher überspringen.

Vorgehensweise

1. Erstellen Sie mit dem Assistenten **New telemetry channel** (Neuer Telemetriekanal) einen Telemetriekanal mit der Bezeichnung SSLCAReqClients, für den eine SSL-Verbindung erforderlich ist. Der Kanal akzeptiert nur authentifizierte Clients.

   Passen Sie Ihre Kanalkonfiguration auf Basis der folgenden Konfigurationszeilengruppe an. Bearbeiten Sie die Telemetrieeigenschaftendatei nicht direkt, sondern verwenden Sie den Assistenten.

```
com.ibm.mq.MQXR.channel/SSLCAReqClients: \
com.ibm.mq.MQXR.Port=8886;\
com.ibm.mq.MQXR.Backlog=4096;\
com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerReqKey.jks;\
com.ibm.mq.MQXR.PassPhrase=password;\
com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

2. Generieren Sie einen von einer Zertifizierungsstelle signierten Schlüssel für den Client zur Authentifizierung des Telemetriekanals.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für den Telemetriekanal in einem neuen Schlüsselspeicher mit dem Namen `CAServerReqKey.jks`:

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerReqKey.jks -storepass password -keypass password
   ```

   Der Schlüsselalgorithmus ist auf RSA gesetzt, weil dies von einigen Zertifizierungsstellen gefordert wird. Der allgemeine Name des Zertifikats muss eindeutig sein, denn einige Zertifizierungsstellen stellen keine Schlüssel mit gleichlautenden allgemeinen Namen aus.

   b) Erstellen Sie eine Zertifikatssignieranforderung in Form einer ASCII-Datei

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerReqKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerReqKey.jks -storepass password -keypass password
   ```

   c) Führen Sie die Software der Zertifizierungsstelle aus oder melden Sie sich bei deren Website an. Fügen Sie die Inhalte von `CAServerReqKey.csr` ein, wenn Sie nach der Datei mit der Zertifikatssignieranforderung gefragt werden.

   d) Die Zertifizierungsstelle gibt ein oder zwei Zertifikate und eine signierte Antwortdatei als ASCII-Dateien zurück. Fügen Sie den Inhalt in zwei oder drei Dateien ein:

   Stammzertifikat
   Fügen Sie dies in `CARoot.cer` ein.

   Zwischenzertifikat
   Fügen Sie dies in `CAInter.cer` ein.

   Server-signierte Antwortdatei
   Fügen Sie dies in `CAServerReq.rsp` ein.

   Der JRE-Zertifikatsspeicher wird in dieser Task nicht verwendet. Wenn Sie ein einziges Stammzertifikat und eine signierte Antwort von der Zertifizierungsstelle empfangen haben, verwenden Sie das Stammzertifikat und die signierte Antwort in den folgenden Schritten. Wenn Sie ein Stamm- und ein Zwischenzertifikat empfangen haben, verwenden Sie das Zwischenzertifikat und die signierte Antwort.

   e) Empfangen Sie die signierte Serverantwort in dem Serverschlüsselspeicher, aus dem Sie die Zertifikatsanforderung gestellt haben.

   Durch das Empfangen der Antwort wird das selbst signierte Zertifikat in ein von der Zertifizierungsstelle signiertes Zertifikat geändert. Wenn Sie sich das Zertifikat vor und nach dem Empfang der Antwort im Schlüsselspeicher anschauen, werden Sie feststellen, dass der Unterzeichner entsprechend geändert wird. Falls nicht, meldet das Schlüsselverwaltungstool einen Fehler. Überprüfen Sie das Zertifikat, bevor Sie es verwenden, und vergewissern Sie sich, dass jetzt die Zertifizierungsstelle als Unterzeichner ausgewiesen wird.

   ```
   Keytool -import -noprompt -alias CAServer -file CAServerReq.rsp
           -keystore CAServerReqKey.jks -storepass password
   ```

   In einigen Schlüsselverwaltungsprogrammen, z. B. in **iKeyman** empfangen Sie Antwortdateien, statt sie zu importieren.

   f) Importieren Sie das Zertifikat der Zertifizierungsstelle in den Client-Truststore.

   Importieren Sie entweder das Zwischenzertifikat, falls Sie zwei Zertifikate von der Zertifizierungsstelle empfangen haben, oder das Stammzertifikat, falls Sie nur ein Zertifikat empfangen haben.

   Entweder:

   ```
   keytool -import -alias CAInter -file CAInter.cer
           -keystore CAClientTrust.jks -storepass password
   ```

Oder:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

3. Generieren Sie einen von einer Zertifizierungsstelle signierten Schlüssel, mit dem der Telemetriekanal Clients authentifizieren kann.

   a) Generieren Sie ein selbst signiertes Schlüsselpaar für die Clients im neuen Schlüsselspeicher `CAClientKey.jks`:

   ```
   Keytool -genkey -noprompt -alias CAClientPrivate -keyalg RSA
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAClientKey.jks -storepass password -keypass password
   ```

   Der Schlüsselalgorithmus ist auf RSA gesetzt, weil dies von einigen Zertifizierungsstellen gefordert wird. Der allgemeine Name des Zertifikats muss eindeutig sein, denn einige Zertifizierungsstellen stellen keine Schlüssel mit gleichlautenden allgemeinen Namen aus.

   b) Erstellen Sie eine Zertifikatssignieranforderung in Form einer ASCII-Datei

   ```
   Keytool -certreq -noprompt -alias CAClient -file CAClientKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAClientKey.jks -storepass password -keypass password
   ```

   c) Führen Sie die Software der Zertifizierungsstelle aus oder melden Sie sich bei deren Website an. Fügen Sie die Inhalte von `CAClientKey.csr` ein, wenn Sie nach der Datei mit der Zertifikatssignieranforderung gefragt werden.

   d) Die Zertifizierungsstelle gibt ein oder zwei Zertifikate und eine signierte Antwortdatei als ASCII-Dateien zurück. Fügen Sie den Inhalt in zwei oder drei Dateien ein:

   Stammzertifikat
   Fügen Sie dies in `CARoot.cer` ein.

   Zwischenzertifikat
   Fügen Sie dies in `CAInter.cer` ein.

   Client-signierte Antwortdatei
   Fügen Sie dies in `CAClient.rsp` ein.

   Der JRE-Zertifikatsspeicher wird in dieser Task nicht verwendet. Wenn Sie ein einziges Stammzertifikat und eine signierte Antwort von der Zertifizierungsstelle empfangen haben, verwenden Sie das Stammzertifikat und die signierte Antwort in den folgenden Schritten. Wenn Sie ein Stamm- und ein Zwischenzertifikat empfangen haben, verwenden Sie das Zwischenzertifikat und die signierte Antwort.

   e) Empfangen Sie die signierte Clientantwort in dem Clientschlüsselspeicher, aus dem Sie die Zertifikatsanforderung gestellt haben.

   Durch das Empfangen der Antwort wird das selbst signierte Zertifikat in ein von der Zertifizierungsstelle signiertes Zertifikat geändert. Wenn Sie sich das Zertifikat vor und nach dem Empfang der Antwort im Schlüsselspeicher anschauen, werden Sie feststellen, dass der Unterzeichner entsprechend geändert wird. Falls nicht, meldet das Schlüsselverwaltungstool einen Fehler. Überprüfen Sie das Zertifikat, bevor Sie es verwenden, und vergewissern Sie sich, dass jetzt die Zertifizierungsstelle als Unterzeichner ausgewiesen wird.

   ```
   Keytool -import -noprompt -alias CAClient -file CAClient.rsp
           -keystore CAClientKey.jks -storepass password
   ```

   In einigen Schlüsselverwaltungsprogrammen, z. B. in **iKeyman** empfangen Sie Antwortdateien, statt sie zu importieren.

   f) Importieren Sie das Zertifikat der Zertifizierungsstelle in den Server-Truststore.

   Importieren Sie entweder das Zwischenzertifikat, falls Sie zwei Zertifikate von der Zertifizierungsstelle empfangen haben, oder das Stammzertifikat, falls Sie nur ein Zertifikat empfangen haben.

Entweder:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAServerReqKey.jks -storepass password
```

Oder:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAServerReqKey.jks -storepass password
```

Erste MQ Telemetry Transport-Publisher-Anwendung unter Verwendung der Programmiersprache C erstellen

Die Schritte zum Erstellen einer MQTT-Client-Publisher-Anwendung werden in Form eines Lernprogramms beschrieben. Dabei wird jede Zeile des C-Codes beschrieben. Am Ende der Task haben Sie einen MQTT-Publisher erstellt.

Vorbereitende Schritte

Die entwickelte Clientanwendung verwendet die MQTT-V3-C-Clientbibliotheken. Zur Veröffentlichung von Nachrichten stellt die Anwendung eine Verbindung zum WebSphere MQ Telemetry-Dämon für Einheiten her. Ein Beispiel für die Kommunikation eines Clients mit WebSphere MQ Telemetry finden Sie im Abschnitt Ersten Publisher erstellen.

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Veröffentlichungsanwendung, `pubsync.c`. Das Programm `pubsync.c` veröffentlicht eine Nachricht mit den Nutzdaten `Hello World!` zum Thema `MQTT Exampleund` wartet auf die Bestätigung, dass die Veröffentlichung an den Dämon zugestellt wurde.

Der Einfachheit halber wird die korrekte Beendigung der Rückkehrcodes einiger verwendeter Funktionen nicht getestet. Im Produktionscode können die Rückkehrcodes geprüft werden, um sicherzugehen, dass das Programm wie erwartet funktioniert. Bei einem unerwarteten Fehler muss die entsprechende Maßnahme ergriffen werden.

Wenn Sie einen Subskribenten für `MQTT Example` einrichten, können Sie überprüfen, ob die Anwendung funktioniert.

Verwenden Sie die ausgewählte C-Entwicklungsumgebung zum Entwickeln, Erstellen und Ausführen des Clients. Sie können den Code auch direkt aus den Beispielen kopieren.

Vorgehensweise

1. Erstellen Sie eine neue leere Quellendatei `pubsync.c`
2. Erstellen Sie eine Datei `settings.h`. Kopieren Sie den Code aus Abbildung 2 in die Datei.

   Alle im Programm verwendeten Parameter sind in `settings.h` definiert. Sie können die Werte überschreiben, indem Sie die Werte in der Datei ändern.
3. In den folgenden Schritten wird der Code erläutert. Führen Sie die Schritte aus oder kopieren Sie den Code aus Abbildung 1 in `pubsync.c`.
4. Fügen Sie die Include-Anweisungen der Headerdatei für die erforderlichen Standardbibliotheken sowie für die Dateien `MQTTClient.h` und `settings.h` hinzu.

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "string.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. Starten Sie die Definition de Funktion `main()`.

```
int main(int argc, char* argv[])
{
```

6. Definieren Sie die im Programm verwendeten lokalen Variablen.

```
MQTTClient client;
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
MQTTClient_message pubmsg;
MQTTClient_deliveryToken token;
int rc;
```

Anmerkung: Für die Funktion `MQTTClient_connect` sind Verbindungsoptionen erforderlich. `MQTTClient_connectOptions_initializer` enthält die entsprechenden Standardoptionen.

7. Erstellen Sie einen Client.

```
MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

- `& client` ist ein Zeiger auf eine Kennung für den neu erstellten Client. Wenn diese Funktion mit dem Rückkehrcode 0 zurückgegeben wird, enthält sie eine Kennung für den neuen Client. In dem Beispiel wird davon ausgegangen, dass der Vorgang erfolgreich ausgeführt werden kann. Im Produktionscode kann die korrekte Beendigung des Fehlercodes getestet werden.
- ADDRESS ist der URI des MQTT-Ports, der vom Dämon für eingehende Clientverbindungsanforderungen überwacht wird.
- `CLIENTID` ist der Name, mit dem der Client auf dem Dämon identifiziert wird. Jeder aktive Client muss einen eindeutigen Namen haben. Wird eine Client-ID auf zwei aktiven Clients doppelt vergeben, wird auf beiden Clients eine Ausnahme ausgelöst und ein Client wird beendet. Anhand des Namens erkennt der Dämon einen Client, der nach einem Verbindungsabbau erneut eine Verbindung herstellt. Informationen hierzu finden Sie im Abschnitt zur Client-ID.
- Mit `MQTTCLIENT_PERSISTENCE_NONE` wird angegeben, dass der Clientstatus im Hauptspeicher gespeichert wird und bei einem Systemfehler verloren geht. Mit `MQTTCLIENT_PERSISTENCE_DEFAULT` wird eine auf dem Dateisystem basierende Persistenz angegeben, die bei Störungen einen gewissen Schutz bietet. Für spezifischere Anwendungen kann der Parameter `MQTTCLIENT_PERSISTENCE_USER` verwendet werden. Dieser bietet Ihnen eine Schnittstelle, über die Sie einen eigenen Persistenzmechanismus implementieren können. Weitere Informationen finden Sie in der API-Dokumentation zu `MQTTClientPersistence.h`. Ob Persistenz erforderlich ist, hängt vom Anwendungsdesign ab. Weitere Informationen hierzu finden Sie im Abschnitt Nachrichtenpersistenz.
- Der TCP/IP-Standardport des Dämons für MQTT lautet 1883. Im Beispiel wird die Standardadresse auf `tcp://localhost:1883` gesetzt.
- Die Nachrichtenverarbeitung beginnt erst, wenn Sie die Funktion `MQTTClient_connect` aufgerufen haben.

8. Verbinden Sie den Client mit dem Dämon.

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
     printf("Failed to connect, return code %d\n", rc);
     exit(-1);
}
```

- Die Funktion `MQTTClient_connect` wird aufgerufen. Dabei werden das Client-Handle und ein Verweis auf die Verbindungsoptionen als Argumente übergeben.
- Der Rückgabecode des Aufrufs `MQTTClient_connect` wird getestet, um sicherzustellen, dass die Verbindungsanforderung erfolgreich war.
- Wenn `MQTTClient_connect` fehlschlägt, wird das Programm mit dem Fehlercode -1 beendet.
- Nachdem die Anwendung eine Verbindung hergestellt hat, können die Veröffentlichungen und Subskriptionen gestartet werden.
- Alle 20 Sekunden wird eine kurze Keepalive-Nachricht gesendet, damit die TCP/IP-Verbindung nicht geschlossen wird. Diese Option wird über `conn_opts.keepAliveInterval` festgelegt.

- Die Sitzung wird gestartet. Da `conn_opts.cleansession` auf 'true' gesetzt ist, wird dabei nicht geprüft, ob aus einer früheren Verbindung verbleibende unvollständige Nachrichten abgeschlossen wurden. Weitere Einzelheiten hierzu finden Sie im Abschnitt Sitzungen bereinigen.
- Für die Verbindung wird keine Nachricht 'Last Will and Testament' erstellt. Weitere Einzelheiten finden Sie im Abschnitt Letzter Wille und Testament.

9. Füllen Sie die Struktur *MQTTClient_message* mit den Daten zur Definition der Nachrichtennutzdaten und der zugehörigen Attribute aus.

```
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
```

   - PAYLOAD (Nutzdaten) ist der Nachrichteninhalt.
   - In dem Beispiel werden Nutzdaten in Form einer Zeichenfolge verwendet, bei MQTT-Nutzdaten handelt es sich jedoch um Bytefeldgruppen. Über die Länge der Zeichenfolge muss die Größe der Nutzdaten angegeben werden.
   - n dem Beispiel wird eine QoS=1-Nachricht veröffentlicht, der Wert ist also entsprechend zu setzen.
   - Das beibehaltene Attribut ist auf (0) gesetzt, da die Nachricht nicht vom Dämon beibehalten werden soll. Weitere Einzelheiten hierzu finden Sie im Abschnitt Ständige Veröffentlichungen.

10. Veröffentlichen Sie die Nachricht.

```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
```

    - In der Veröffentlichungsfunktion sind der Client, das Thema und die an den Dämon zu sendenden Nutzdaten angegeben.
    - `TOPIC` ist in `settings.h` als `MQTT Example`definiert.
    - Der Funktion wird auch ein Zeiger auf ein *MQTTClient_deliveryToken* übergeben. Dieser Zeiger wird bei der Funktionsrückgabe mit einem Token zur Darstellung der Nachricht belegt.
    - Die Nachricht wird nun sicher an den MQTT-Client übertragen, die Übertragung an den Dämon erfolgt jedoch noch nicht. Hat die Nachricht die Servicequalität QoS=1 oder 2, wird sie lokal gespeichert, für den Fall, dass es auf dem Client zu einem Fehler kommt, bevor die Zustellung abgeschlossen ist.
    - Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden kann.

11. Warten Sie auf die Empfangsbestätigung des Servers.

```
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

    - Im Beispiel `pubsync.c` wird auf eine Empfangsbestätigung des Servers gewartet, mit der die Nachrichtenzustellung bestätigt wird.
    - Über die Client- und Tokenargumente wird die spezielle Nachricht bezeichnet, auf deren Abschluss das Programm wartet.
    - Über den Parameter `TIMEOUT` wird festgelegt, wie lange das Programm auf den Abschluss der Nachrichtenübergabe wartet. In der Task Unter Verwendung der Programmiersprache C asynchronen Publisher für MQ Telemetry Transport erstellen wird erläutert, wie mithilfe von Callback-Funktionen ohne Wartezeit Empfangsbestätigungen erhalten werden können.
    - Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden kann.

12. Trennen Sie die Verbindung des Clients zum Dämon.

```
MQTTClient_disconnect(client, 10000);
```

- Die Verbindung des Clients zum Server wird getrennt und der Client wartet auf Callback-Funktionen (in diesem Beispiel nicht verwendet) zu unvollständigen Nachrichten, die noch beendet werden müssen.
- Mit dem zweiten Argument wird ein Quiesce-Zeitlimit in Millisekunden angegeben. In dem Beispiel wird maximal 10 Sekunden gewartet, damit andere Arbeitsgänge beendet werden können, bevor die Verbindung getrennt wird.
- Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden muss.

13. Geben Sie den vom Client belegten Speicher frei und beenden Sie das Programm.

```
MQTTClient_destroy(&client);
}
```

Ergebnisse

Erstellen Sie zum Anzeigen der von diesem Client gesendeten Veröffentlichungen einen Subskribenten für das Thema `MQTT Example`. Weitere Einzelheiten hierzu finden Sie im Abschnitt Unter Verwendung der Programmiersprache C Subskribenten für MQ Telemetry Transport erstellen.

Beispiel

In Abbildung 1 ist der gesamte im Abschnitt Vorgehensweise beschriebene Code aufgeführt. In der in Abbildung 2 dargestellten Datei `settings.h` können die in `pubsync.c` verwendeten Standardparameter geändert werden.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"

int pubsync_main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID,  MQTTCLIENT_PERSISTENCE_NONE, NULL);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
          printf("Failed to connect, return code %d\n", rc);
         exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for up to %d seconds for publication of %s\n"
          "on topic %s for client with ClientID: %s\n",
          TIMEOUT/1000, PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Abbildung 111. `pubsync.c`

```
#define ADDRESS   "tcp://localhost:1883"
#define CLIENTID  "ExampleClientPub"
#define TOPIC     "MQTT Example"
#define PAYLOAD   "Hello World!"
#define QOS      1
#define TIMEOUT   10000L
```

Abbildung 112. `settings.h`

Unter Verwendung der Programmiersprache C asynchronen Publisher für MQ Telemetry Transport erstellen

Die Schritte zum Erstellen einer asynchronen Publisher-Anwendung für MQTT-Clients werden in Form eines Lernprogramms beschrieben. Dabei wird jede Zeile des C-Codes beschrieben. Am Ende der Task werden Sie einen asynchronen MQTT-Publisher erstellt haben.

Bei dieser Task führen Sie ein Lernprogramm aus, um Ihre erste Publisher-Anwendung zu ändern. Mit den Änderungen kann die Anwendung Veröffentlichungen senden, ohne auf Empfangsbestätigungen warten zu müssen. Die Empfangsbestätigungen werden von einer von Ihnen zu erstellenden Callback-Funktion empfangen.

Vorbereitende Schritte

Die entwickelte Clientanwendung verwendet die MQTT-V3-C-Clientbibliotheken. Zur Veröffentlichung von Nachrichten stellt die Anwendung eine Verbindung zum WebSphere MQ Telemetry-Dämon für Einheiten her. Ein Beispiel für die Kommunikation eines Clients mit WebSphere MQ Telemetry finden Sie im Abschnitt Ersten Publisher erstellen.

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Veröffentlichungsanwendung,`pubasync.c`. Das Programm `pubasync.c` veröffentlicht eine Nachricht mit den Nutzdaten `Hello World!` zum Thema `MQTT Example`, ohne auf die Bestätigung zu warten, dass die Veröffentlichung an den Dämon zugestellt wurde. Die Empfangsbestätigungen werden in einer Callback-Funktion mit dem Namen `MQTTClient_deliveryComplete` empfangen.

Der Einfachheit halber wird die korrekte Beendigung der Rückkehrcodes einiger verwendeter Funktionen nicht getestet. Im Produktionscode können die Rückkehrcodes geprüft werden, um sicherzugehen, dass das Programm wie erwartet funktioniert. Bei einem unerwarteten Fehler muss die entsprechende Maßnahme ergriffen werden.

Durch das Einrichten eines Subskribenten für `MQTT Example` können Sie prüfen, ob die Anwendung funktioniert.

Verwenden Sie die ausgewählte C-Entwicklungsumgebung zum Entwickeln, Erstellen und Ausführen des Clients.

Mit den unter Vorgehensweise erläuterten Schritten wird die Anwendung `pubsync.c` aus dem Abschnitt „Erste MQ Telemetry Transport-Publisher-Anwendung unter Verwendung der Programmiersprache C erstellen” auf Seite 543 geändert. Sie können den Code auch direkt aus den Beispielen kopieren.

Vorgehensweise

1. Erstellen Sie eine neue leere Quellendatei, `callback.h`.
2. Kopieren Sie den Code aus Abbildung 2 in die Datei.
   - In der Datei `callback.h` sind die drei für den asynchronen Clientbetrieb erforderlichen Callback-Methoden deklariert.
   - Es wird auch eine Variable *deliveredtoken* deklariert. Der Zugriff auf diese Variable erfolgt über das Hauptprogramm und den Callback an verschiedenen Ausführungs-Threads. Sie wird daher als flüchtig deklariert. Achten Sie bei der Verwendung von Callbacks darauf, dass ein threadsicherer Zugriff auf die betreffenden Variablen erfolgt.
3. Erstellen Sie eine neue leere Quellendatei, `callback.c`.
4. Kopieren Sie den Code aus Abbildung 3 in die Datei.
   - Mit der Datei `callback.c` werden die drei Callback-Methoden implementiert, die vom Client für den asynchronen Betrieb verwendet werden: `delivered`, `msgarrvd` und `connlost`.

5. Fügen Sie im Anschluss an die anderen include-Anweisungen in der Datei pubasync.c eine include-Anweisung für callback.h hinzu.

```
#include "callback.h"
```

6. Kopieren Sie die Inhalte der Datei pubsync.c in eine neue Datei pubasync.c..
7. Definieren Sie unmittelbar vor dem Funktionsaufruf MQTTClient_connect in der Datei pubasync.c die Callback-Methoden für den Client.

```
MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
```

   - Es müssen drei Callback-Funktionen angegeben werden. Diese Funktionen werden in callback.c implementiert.
   - MQTTClient_messageArrived wird aufgerufen, wenn aufgrund einer entsprechenden Subskription eine Nachricht an den Client gesendet wird. Dabei muss 'true' zurückgegeben werden, wenn die erhaltene Nachricht erfolgreich von der Clientanwendung empfangen wurde. Wird 'false' zurückgegeben, weist dies den Client darauf hin, dass die Anwendung beim Empfang der Nachricht ein Problem hatte.
   - MQTTClient_connectionLost wird aufgerufen, wenn die Clientverbindung zum Server unterbrochen wird.
   - MQTTClient_deliveryComplete wird aufgerufen, wenn eine QoS1- oder QoS2-Nachricht empfangen und vom Server bestätigt wurde. Bei QoS0-Nachrichten wird diese Funktion nicht aufgerufen. In dem Beispiel speichert die Funktion das Token aus der zugestellten Nachricht unter *deliveredtoken* und gibt damit an, dass eine Nachricht angekommen ist.
   - MQTTClient_setCallbacks muss aufgerufen werden, solange keine Verbindung des Clients zum Server besteht.
   - Mit dem zweiten Argument können den Callback-Funktionen Kontextinformationen übergeben werden. In dem Beispiel wird es nicht verwendet, es ist also auf NULL gesetzt.
8. Inaktivieren Sie unmittelbar vor dem Aufruf von MQTTClient_publishMessage die Variable *deliveredtoken*. Bei Empfang eines Tokens wird die Variable *deliveredtoken* durch MQTTClient_deliveryComplete erneut aktiviert.

```
deliveredtoken = 0;
```

9. Entfernen Sie den Aufruf MQTTClient_waitForCompletion sowie die darauf folgende Anweisung printf und ersetzen Sie sie durch eine Schleife, die auf eine Übereinstimmung mit dem ursprünglichen Token und dem im Rückruf empfangenen Token wartet.

```
while(deliveredtoken != token);
```

   Hierbei handelt es sich um ein Beispiel, das für eine Reihe von Situationen, die in der Produktionscodeentwicklung berücksichtigt werden müssen, nicht ausreichend ist. Dazu zählen u. a. folgende Situationen:

   - Falls die Zustellung nicht abgeschlossen wird, kann ein Zeitlimit implementiert werden.
   - Es können auch mehrere Nachrichten unvollständig sein. In dem Musterprogramm kann immer nur jeweils ein Zustellungstoken geprüft werden.
10. Trennen Sie die Verbindung des Clients zum Dämon.

```
MQTTClient_disconnect(client, 10000);
```

    - Die Verbindung des Clients zum Server wird getrennt und der Client wartet auf Callback-Funktionen zu unvollständigen Nachrichten, die noch beendet werden müssen.
    - Mit dem zweiten Argument wird ein Quiesce-Zeitlimit in Millisekunden angegeben. In dem Beispiel wird maximal 10 Sekunden gewartet, damit andere Arbeitsgänge beendet werden können, bevor die Verbindung getrennt wird.

- Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden muss.

11. Geben Sie den vom Client belegten Speicher frei und beenden Sie das Programm.

```
MQTTClient_destroy(&client);
}
```

Ergebnisse

Erstellen Sie zum Anzeigen der von diesem Client gesendeten Veröffentlichung einen Subskribenten für das Thema `MQTT Example`. Weitere Einzelheiten hierzu finden Sie im Abschnitt Subskribenten für MQ Telemetry Transport erstellen.

Beispiel

In den Dateien `pubasync.c`, `callbacks.c` und `callbacks.h` ist der gesamte im Abschnitt Vorgehensweise beschriebene Code aufgeführt.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    deliveredtoken = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for publication of %s\n"
           "on topic %s for client with ClientID: %s\n", PAYLOAD, TOPIC, CLIENTID);
    while(deliveredtoken != token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Abbildung 113. `pubasync.c`

```
MQTTClient_deliveryComplete delivered;
MQTTClient_messageArrived msgarrvd;
MQTTClient_connectionLost connlost;

extern volatile MQTTClient_deliveryToken deliveredtoken;
```

Abbildung 114. `callback.h`

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt)
{
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
    }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Abbildung 115. `callback.c`

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientPub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Abbildung 116. `settings.h`

Unter Verwendung der Programmiersprache C Subskribenten für MQ Telemetry Transport erstellen

Die Schritte zum Erstellen einer MQTT-Client-Subskribentenanwendung werden in Form eines Lernprogramms beschrieben. Dabei wird jede Zeile des C-Codes beschrieben. Am Ende der Task werden Sie einen MQTT-Subskribenten erstellt haben.

Vorbereitende Schritte

Die entwickelte Clientanwendung verwendet die MQTT-V3-C-Clientbibliotheken. Zur Veröffentlichung von Nachrichten stellt die Anwendung eine Verbindung zum WebSphere MQ Telemetry-Dämon für Einheiten her. Ein Beispiel für die Kommunikation eines Clients mit WebSphere MQ Telemetry finden Sie im Abschnitt Ersten Publisher erstellen.

Informationen zu diesem Vorgang

Bei dem Beispiel handelt es sich um eine Subskribentenanwendung, `subscribe.c`. Das Programm `subscribe.c` subskribiert das Thema `MQTT Example` und wartet auf Veröffentlichungen, die mit der Subskription übereinstimmen, bis der Benutzer das Programm beendet.

Ein Subskribent erstellt eine Subskription für ein Thema und wartet auf Nachrichten, die dem Subskriptionsthema entsprechen. Nachrichten, die veröffentlicht werden, während die Clientverbindung getrennt ist und die einer zuvor vom Client erstellten Subskription entsprechen, können empfangen werden, wenn die

Clientverbindung wiederhergestellt wird. Der WebSphere MQ Telemetry-Service (MQXR) oder der Dämon für Einheiten erkennt einen Client, für den früher anhand der Client-ID eine Verbindung hergestellt wurde. Weitere Informationen zu diesem Thema finden Sie im Abschnitt zur Client-ID. Mit dem booleschen Attribut `MQTTClient_connectOptions.cleansession` wird gesteuert, ob zuvor gesendete Veröffentlichungen empfangen werden oder nicht. Weitere Informationen finden Sie im Thema „Sitzungen bereinigen" auf Seite 560.

Der Einfachheit halber wird die korrekte Beendigung der Rückkehrcodes einiger verwendeter Funktionen nicht getestet. Im Produktionscode können die Rückkehrcodes geprüft werden, um sicherzugehen, dass das Programm wie erwartet funktioniert. Bei einem unerwarteten Fehler kann die entsprechende Maßnahme ergriffen werden.

Sie können die zuvor erläuterten Musterveröffentlichungsprogramme verwenden, um entsprechende Veröffentlichungen an den WebSphere MQ Telemetry-Dämon für Einheiten zu senden. Alternativ dazu können Sie auch über den WebSphere MQ-Explorer Testveröffentlichungen zum Thema `MQTT Example` erstellen, wenn Sie den Client mit einem WebSphere MQ Telemetry-Kanal verbinden wollen.

Bei den Anweisungen im Abschnitt Vorgehensweise wird davon ausgegangen, dass die Dateien `callback.c`, `callback.h` und `settings.h` in einer der früheren Tasks erstellt wurden.

Verwenden Sie die ausgewählte C-Entwicklungsumgebung zum Entwickeln, Erstellen und Ausführen des Clients. Sie können den Code auch direkt aus den Beispielen kopieren.

Vorgehensweise

1. Erstellen Sie für dieses Beispiel eine Kopie der Datei `settings.h` und ändern Sie die Definitionsanweisung für *CLIENTID* folgendermaßen:

   ```
   #define CLIENTID "ExampleClientSub"
   ```

   - Falls zwei Clients mit derselben ID versuchen, eine Verbindung zu einem einzigen Server herzustellen, wird einer der beiden Clients zwangsweise getrennt. Im Allgemeinen ist der neue Versuch des Verbindungsaufbaus erfolgreich und die ältere Verbindung wird getrennt.
   - Durch das Ändern der `ClientID` können Sie die zuvor entwickelten Veröffentlichungsbeispiele zum Senden von Nachrichten an diesen Subskribenten verwenden.
2. Erstellen Sie eine neue leere Quellendatei, `subscribe.c`.
3. In den folgenden Schritten wird der Code erläutert. Führen Sie die Schritte aus oder kopieren Sie den Code aus Abbildung 117 auf Seite 555 in die Datei `subscribe.c`.
4. Fügen Sie die Include-Anweisungen der Headerdatei für die erforderlichen Standardbibliotheken sowie für die Dateien `MQTTClient.h` und `settings.h` hinzu.

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```

5. Starten Sie die Definition de Funktion `main()`.

   ```
   int main(int argc, char* argv[]) {
   ```

6. Definieren Sie die im Programm verwendeten lokalen Variablen.

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   Für die Funktion `MQTTClient_connect` sind Verbindungsoptionen erforderlich. `MQTTClient_connectOptions_initializer` enthält die entsprechenden Standardoptionen.
7. Erstellen Sie einen Client.

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

- `& client` ist ein Zeiger auf eine Kennung für den neu erstellten Client. Wenn diese Funktion mit dem Rückkehrcode 0 zurückgegeben wird, enthält der Verweis eine Kennung für den neuen Client. In dem Beispiel wird davon ausgegangen, dass der Vorgang erfolgreich ausgeführt werden kann. Der Fehlercode kann auf die korrekte Beendigung im Produktionscode getestet werden.
- ADDRESS ist der URI des MQTT-Ports, der vom Dämon für eingehende Clientverbindungsanforderungen überwacht wird.
- `CLIENTID` ist der Name, mit dem der Client auf dem Dämon identifiziert wird. Jeder aktive Client muss einen eindeutigen Namen haben. Wird eine Client-ID auf zwei aktiven Clients doppelt vergeben, wird auf beiden Clients eine Ausnahme ausgelöst und ein Client wird beendet. Anhand des Namens erkennt der Dämon einen Client, der nach einem Verbindungsabbau erneut eine Verbindung herstellt. Informationen hierzu finden Sie im Abschnitt zur Client-ID.
- Mit `MQTTCLIENT_PERSISTENCE_NONE` wird angegeben, dass der Clientstatus im Hauptspeicher gespeichert wird und bei einem Systemfehler verloren geht. Mit `MQTTCLIENT_PERSISTENCE#_DEFAULT` wird eine auf dem Dateisystem basierende Persistenz angegeben, die bei Störungen einen gewissen Schutz bietet. Für spezifischere Anwendungen kann der Parameter `MQTTCLIENT_PERSISTENCE_USER` verwendet werden. Dieser bietet Ihnen eine Schnittstelle, über die Sie einen eigenen Persistenzmechanismus implementieren können. Ob Persistenz erforderlich ist, hängt vom Anwendungsdesign ab. Weitere Informationen hierzu finden Sie im Abschnitt Nachrichtenpersistenz.
- Der TCP/IP-Standardport des Dämons für MQTT lautet 1883. Im Beispiel wird die Standardadresse auf `tcp://localhost:1883` gesetzt.
- Die Nachrichtenverarbeitung beginnt erst, wenn Sie die Funktion `MQTTClient_connect` aufgerufen haben.

8. Verbinden Sie den Client mit dem Dämon.

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
    printf("Failed to connect, return code %d\n", rc);
    exit(-1);
}
```

   - Die Funktion `MQTTClient_connect` wird aufgerufen. Dabei werden das Client-Handle und ein Verweis auf die Verbindungsoptionen als Argumente übergeben.
   - Der Rückgabecode des Aufrufs `MQTTClient_connect` wird getestet, um sicherzustellen, dass die Verbindungsanforderung erfolgreich war.
   - Wenn der Verbindungsaufruf fehlschlägt, wird das Programm mit dem Fehlercode -1 beendet.
   - Nachdem die Anwendung eine Verbindung hergestellt hat, können die Veröffentlichungen und Subskriptionen gestartet werden.
   - Alle 20 Sekunden wird eine kurze Keepalive-Nachricht gesendet, damit die TCP/IP-Verbindung nicht geschlossen wird. Diese Option wird über `conn_opts.keepAliveInterval` festgelegt.
   - Die Sitzung wird gestartet. Da *conn_opts.cleansession* auf 'true' gesetzt ist, wird dabei nicht geprüft, ob aus einer früheren Verbindung verbleibende unvollständige Nachrichten abgeschlossen wurden. Weitere Einzelheiten hierzu finden Sie im Abschnitt Sitzungen bereinigen.
   - Für die Verbindung wird keine Nachricht 'Last Will and Testament' erstellt. Weitere Einzelheiten finden Sie im Abschnitt Letzter Wille und Testament.

9. Subskribieren Sie das Thema.

```
MQTTClient_subscribe(client, TOPIC, QOS);
```

   - Verwenden Sie die Funktion `MQTTClient_subscribe` , um die Clientanwendung für das ausgewählte Thema zu subskribieren. Der Themenname kann Platzhalterzeichen enthalten. Weitere Informationen finden Sie im Thema „Themenzeichenfolgen und Themenfilter in MQTT-Clients” auf Seite 576.
   - Über die QoS-Einstellung wird die höchste Servicequalität bestimmt, die auf an diesen Subskribenten gesendete Nachrichten angewendet ist. Wenn der Server Nachrichten sendet, gilt der jeweils geringere Wert von dieser Einstellung und der QoS-Einstellung der ursprünglichen Nachricht.

- Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden kann.

10. Warten Sie in einer Schleife, bis der Benutzer über die Tastatur das Zeichen ‘Q' eingibt.

```
do {
    ch = getchar();
    } while(ch!='Q' && ch != 'q');
```

Das Programm wartet nun auf ankommende Nachrichten. In diesem Beispiel erfolgt die gesamte Nachrichtenbehandlung in der Callback-Funktion `MQTTClient_messageArrived`. Weitere Informationen finden Sie im Thema „Nachrichten empfangen” auf Seite 553.

11. Trennen Sie die Verbindung des Clients zum Dämon.

```
MQTTClient_disconnect(client, 10000);
```

- Die Verbindung des Clients zum Server wird getrennt und der Client wartet auf Callback-Funktionen (in diesem Beispiel nicht verwendet) zu unvollständigen Nachrichten, die noch beendet werden müssen.
- Mit dem zweiten Argument wird ein Quiesce-Zeitlimit in Millisekunden angegeben. In dem Beispiel wird maximal 10 Sekunden gewartet, damit andere Arbeitsgänge beendet werden können, bevor die Verbindung getrennt wird.
- Diese Funktion gibt einen Fehlercode zurück, dessen korrekte Beendigung im Produktionscode getestet werden kann.

12. Geben Sie den vom Client belegten Speicher frei und beenden Sie das Programm.

```
MQTTClient_destroy(&client);
}
```

Nachrichten empfangen

Informationen zu diesem Vorgang

Wenn Nachrichten vom Server ankommen, wird die Funktion `MQTTClient_messageArrived` gestartet. In den folgenden Schritten wird der Code erläutert.

Vorgehensweise

1. Starten Sie die Definition der Callback-Funktion. Diese Definition muss der Funktionsvorlage `MQTTClient_messageArrived` entsprechen.

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
```

- Der Parameter `context` bietet Zugriff auf den Kontext, der beim Aufruf der Funktion `MQTTClient_setCallbacks` an die Clientbibliothek übergeben wurde. Diese Funktion wird in dem Beispiel nicht verwendet.
- `topicName` ist ein Verweis auf das Thema, zu dem die empfangene Nachricht veröffentlicht wurde. Falls bei der Subskription Platzhalterzeichen verwendet wurden, wird über diesen Parameter das spezielle Thema der Nachricht angegeben.
- `topicLen` gibt die Länge der Themenzeichenfolge an. Diese Option wird für Benutzer bereitgestellt, die in ihre Themenzeichenfolgen Nullzeichen integrieren müssen.
- `message` ist ein Verweis auf die Struktur `MQTTClient_message` mit den Nachrichtennutzdaten und Attributen.

2. Definieren Sie die verwendeten lokalen Variablen.

```
int i;
char* payloadptr;
```

In dem Beispiel werden diese Variablen zur Ausgabe der Nutzdaten durch Iteration verwendet.

3. Drucken Sie eine Nachricht mit Thema und Nachrichtennutzdaten aus.

```
printf("Message arrived\n");
printf("     topic: %s\n",topicName);
printf("   message: ");
payloadptr = message->payload;
for(i=0; i<message->payloadlen; i++){
    putchar(*payloadptr++);
    }
putchar('\n');
```

   - In dem Beispiel wird davon ausgegangen, dass es sich bei den empfangenen Nutzdaten um eine Folge druckbarer Zeichen handelt.
   - Bei MQTT-Nutzdaten handelt es sich um eine Bytefeldgruppe. Für die Interpretation ihrer Bedeutung ist die Anwendung zuständig.
4. Geben Sie den zum Speichern der Nachricht verwendeten Speicher frei.

```
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
```

   - In dem Beispiel erfolgt die gesamte Nachrichtenbehandlung in der Callback-Funktion.
   - Stellen Sie sicher, dass die Callback-Funktionen kurz sind und die Steuerung so bald wie möglich an den aufrufenden Thread zurückgeben.
   - Der Nachrichtenzeiger wird zur Bearbeitung im Hauptteil des Programms übergeben.
   - Bei Beendigung der Verarbeitung muss das Hauptprogramm den von der Nachricht belegten Speicher freigeben. `MQTTClient_freeMessage()` ist eine Vereinfachungsfunktion, die die beiden Speicherblöcke an das System zurückgibt, in denen die Struktur *MQTTClient_message* und die Nachrichtennutzdaten gespeichert sind. Der `topicName` zugeordnete Speicher muss wie gezeigt separat freigegeben werden.
5. Geben Sie den Wert 'true' zurück, wenn die Nachricht erfolgreich von der Callback-Funktion bearbeitet wurde.

```
     return 1;
}
```

   - Wenn der Wert 'true' zurückgegeben wird, bedeutet dies, dass die Nachricht von der Clientbibliothek als erfolgreich zugestellt behandelt werden kann.
   - Kann die Callback-Funktion die Nachricht nicht ordnungsgemäß verarbeiten, wird der Wert 'false' zurückgegeben. Werden beispielsweise von der Callback-Funktion Nachrichten in eine Warteschlange zur Verarbeitung durch das Hauptprogramm eingereiht und ist diese Warteschlange voll, müsste korrekterweise 'false' zurückgegeben werden.
   - Bei QoS1- und QoS2-Nachrichten bedeutet die Rückgabe von 'false', dass die Nachricht noch nicht zugestellt wurde und weitere Zustellungsversuche unternommen werden.

Beispielcode

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc;
    int ch;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
        }
    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

    MQTTClient_subscribe(client, TOPIC, QOS);
    do {
        ch = getchar();
        } while(ch!='Q' && ch != 'q');
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Abbildung 117. `subscriber.c`

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt) {
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
        }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
void connlost(void *context, char *cause) {
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Abbildung 118. `callback.h`

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientSub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Abbildung 119. `settings.h`

Konzepte zur Programmierung von MQTT-Clients

Die in diesem Abschnitt beschriebenen Konzepte erleichtern Ihnen das Verständnis für die Java-, JavaScript und C-Clientbibliotheken für Version 3.1 des MQTT protocols. Die Konzepte ergänzen die API-Dokumentation, die zu den Clientbibliotheken gehört.

`com.ibm.micro.client.mqttv3` enthält die Klassen, die die öffentlichen Methoden für die Clientbibliotheken für das Protokoll MQTT Version 3.1 bereitstellen. Eine Version des Pakets `com.ibm.micro.client.mqttv3` und der zugehörigen Pakete, mit denen das Protokoll für Java SE und ME implementiert wird, wird durch die Installation von IBM WebSphere MQ Telemetry bereitgestellt. Informationen zum Abrufen der neuesten Version der MQTT -Clientbibliotheken (Java, JavaScript und zum Anzeigen oder Herunterladen der API-Dokumentation finden Sie unter "MQTT-Clientprogrammierungsreferenz".

Zum Entwickeln und Ausführen eines MQTT-Clients müssen Sie diese Pakete 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 von Version 3.1 des MQTT protocols. Sie können Ihre eigenen Clients in unterschiedlichen Sprachen implementieren, die für unterschiedliche Einheitenplattformen geeignet sind. Weitere Informationen hierzu finden Sie im Abschnitt zum 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 WebSphere MQ Telemetry Service verbunden ist.

Callbacks und Synchronisierung in MQTT-Clientanwendungen

Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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

Die `MqttCallback`-Schnittstelle verfügt über drei Callback-Methoden; eine Beispielimplementierung finden Sie unter Callback.java.

`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(MqttDeliveryToken token)`

`deliveryComplete` wird vom MQTT-Client aufgerufen, um ein Zustellungstoken wieder an die Clientanwendung zu übergeben; siehe „Zustellungstoken” auf Seite 563. Mithilfe des Zustellungstokens kann der Callback mit der Methode `token.getMessage` auf die veröffentlichte Nachricht zugreifen.

Wenn der Anwendungscallback die Steuerung nach seinem Aufruf durch die Methode `deliveryComplete` an den MQTT-Client zurückgibt, ist die Übergabe abgeschlossen. Bis zum Abschluss der Übergabe werden Nachrichten mit der Servicequalität QoS von 1 oder 2 von der Persistenzklasse beibehalten.

Der Anruf von `deliveryComplete` stellt einen Synchronisationspunkt zwischen der Anwendung und der Persistenzklasse dar. Die Methode `deliveryComplete` wird für dieselbe Nachricht nie zweimal aufgerufen.

Wenn der Anwendungscallback die Steuerung nach seinem Aufruf durch die Methode `deliveryComplete` an den MQTT-Client zurückgibt, ruft der Client `MqttClientPersistence.remove` für Nachrichten mit der Servicequalität (QoS) 1 oder 2 auf. `MqttClientPersistence.remove` löscht die lokal gespeicherte Kopie der veröffentlichten Nachricht.

Aus Sicht der Transaktionsverarbeitung ist der Aufruf von `deliveryComplete` eine einphasige Transaktion, mit der die Übergabe festgeschrieben wird. Falls während des Callbacks ein Verarbeitungsfehler auftritt, wird `MqttClientPersistence.remove` beim Neustart des Clients erneut aufgerufen, um die lokale Kopie der veröffentlichten Nachricht zu löschen. Der Callback wird nicht erneut aufgerufen. Wenn Sie den Callback zum Speichern eines Protokolls der zugestellten Nachrichten verwenden, können Sie das Protokoll nicht mit dem MQTT-Client synchronisieren. Um ein Protokoll auf zuverlässige Weise zu speichern, müssen Sie das Protokoll in der Klasse `MqttClientPersistence` aktualisieren.

Das Zustellungstoken und die Nachricht werden vom Hauptanwendungsthread und vom MQTT-Client referenziert. Der MQTT-Client nimmt die Referenz zum `MqttMessage`-Objekt zurück, sobald die Übergabe abgeschlossen ist, und das Zustellungstokenobjekt, wenn der Client die Verbindung trennt. Das `MqttMessage`-Objekt kann fehlerhafte Daten, die nach Abschluss der Übergabe erfasst werden, enthalten, wenn die Clientanwendung die Referenz löscht. Das Zustellungstoken kann fehlerhafte Daten enthalten, die nach der Unterbrechung der Sitzung erfasst werden.

Nach dem Veröffentlichen einer Nachricht können Sie die Attribute `MqttDeliveryToken` und `MqttMessage` abrufen. 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 sich der Client mit derselben `Client-ID` wieder mit der vorherigen Sitzung verbindet; siehe „Sitzungen bereinigen" auf Seite 560. Die MQTT-Clientanwendung muss `MqttClient.CleanSession` für die vorherige Sitzung auf `false` und in einer neuen Sitzung ebenfalls auf `false` setzen. Der MQTT-Client erstellt neue Zustellungstokens und Nachrichtenobjekte in der neuen Sitzung für anstehende Übergaben. Er stellt die Objekte mithilfe der Klasse `MqttClientPersistence` wieder her. Falls der Anwendungsclient noch Referenzen auf die alten Zustellungstokens und Nachrichten enthält, löschen Sie diese Referenzen. Der Anwendungscallback wird in der neuen Sitzung für alle Übergaben aufgerufen, die in der früheren Sitzung eingeleitet wurden und in dieser Sitzung abgeschlossen werden.

Der Anwendungscallback wird nach Herstellung der Anwendungsclientverbindung aufgerufen, wenn eine anstehende Übergabe abgeschlossen ist. Bevor der Anwendungsclient die Verbindung herstellt, kann er anstehende Übergaben mit der Methode `MqttClient.getPendingDeliveryTokens` abrufen.

Beachten Sie, dass das Nachrichtenobjekt, das veröffentlicht wird, und dessen Bytefeldgruppe mit Nutzdaten ursprünglich von der Clientanwendung erstellt wurden. Der MQTT-Client verweist auf diese Objekte. Das Nachrichtenobjekt, das vom Zustellungstoken in der Methode `token.getMessage` zurückgegeben wird, ist nicht notwendigerweise dasselbe Nachrichtenobjekt, das vom Client erstellt wurde. Wenn eine neue MQTT-Clientinstanz das Zustellungstoken erneut erstellt, erstellt die Klasse `MqttClientPersistence` das Objekt `MqttMessage` erneut. Aus Konsistenzgründen gibt `token.getMessage` den Wert `null` zurück, wenn `token.isCompleted` auf `true` gesetzt ist, unabhängig davon, ob das Nachrichtenobjekt vom Anwendungsclient oder von der Klasse `MqttClientPersistence` erstellt wurde.

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` wird aufgerufen, wenn eine Veröffentlichung für den Client eintrifft, die mit einem Subskriptionsthema übereinstimmt. `topic` gibt das Veröffentlichungsthema an, nicht den Subskriptionsfilter. Die beiden können unterschiedlich sein, wenn der Filter Platzhalterzeichen enthält.

Wenn das Thema mit mehreren vom Client erstellten Subskriptionen übereinstimmt, empfängt der Client mehrere Kopien der Veröffentlichung. Wenn ein Client eine Veröffentlichung für ein Thema durchführt, das er selbst subskribiert hat, empfängt er eine Kopie seiner eigenen Veröffentlichung.

Wenn eine Nachricht mit der Servicequalität QoS von 1 oder 2 gesendet wird, wird sie von der Klasse `MqttClientPersistence` gespeichert, bevor der MQTT-Client `messageArrived` aufruft. `messageArrived` verhält sich wie `deliveryComplete`: Dieses Element wird nur einmal für eine Veröffentlichung aufgerufen und die lokale Kopie der Veröffentlichung wird von `MqttClientPersistence.remove` entfernt, wenn `messageArrived` an den MQTT-Client zurückgegeben wird. Der MQTT-Client löscht seine Verweise auf das Thema und die Nachricht, wenn `messageArrived` an den MQTT-Client zurückgegeben wird. Die Themen- und Nachrichtenobjekte enthalten fehlerhafte Daten, wenn der Anwendungsclient keine Referenzen auf die Objekte beibehalten hat.

Synchronisation von Callbacks, Threading und Clientanwendungen

Der MQTT-Client ruft eine Callback-Methode nicht im Hauptanwendungsthread, sondern in einem separaten Thread auf. Die Clientanwendung erstellt keinen Thread für den Callback, sondern dieser wird vom MQTT-Client erstellt.

Der MQTT-Client synchronisiert Callback-Methoden. Es wird immer nur eine Instanz der Callback-Methode ausgeführt. Dank der Synchronisation ist es einfach, ein Objekt zu aktualisieren, das mitzählt, welche Veröffentlichungen zugestellt wurden. Da immer nur eine Instanz von `MqttCallback.deliveryComplete` ausgeführt wird, ist die Aktualisierung des Veröffentlichungszählers auch ohne weitere Synchronisation ein zuverlässiger Vorgang. Es trifft außerdem immer nur eine Veröffentlichung ein. Ihr Code in der Methode `messageArrived` kann ein Objekt aktualisieren, ohne es zu synchronisieren. Wenn Sie sich in einem anderen Thread auf den Zähler oder das Objekt, das aktualisiert wird, beziehen, synchronisieren Sie den Zähler oder das Objekt.

Das Zustellungstoken stellt einen Synchronisationsmechanismus zwischen dem Hauptanwendungsthread und der Übergabe einer Veröffentlichung bereit. Die Methode `token.waitForCompletion` wartet, bis die Übergabe einer bestimmten Veröffentlichung abgeschlossen ist oder bis ein optionales Zeitlimit überschritten wird. Sie können `token.waitForCompletion` auf verschiedene Weise verwenden, um eine Veröffentlichung nach der anderen zu verarbeiten:

1. Halten Sie den Anwendungsclient an, bis die Übergabe der Veröffentlichung abgeschlossen ist (siehe Abbildung 88 auf Seite 510).
2. Führen Sie eine Synchronisation mit der Methode `MqttCallback.deliveryComplete` durch. `token.waitForCompletion` wird erst fortgesetzt, wenn `MqttCallback.deliveryComplete` an den MQTT-Client zurückgegeben wird. 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

Tabelle 70 auf Seite 559 beschreibt die Methoden im MQTT -Java-Client, die eine Anfrage 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 70. 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.
`MqttDeliveryToken.waitForCompletion`	Wartet auf die Rückgabe des Zustellungstokens.	Unendlich oder ein mit einem Parameter festgelegter Wert.

Zugehörige Konzepte

Sitzungen bereinigen

Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"

Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients

Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Sitzungen bereinigen

Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Wenn Sie mithilfe der Methode `MqttClient.connect` eine Verbindung für eine MQTT-Clientanwendung herstellen, identifiziert der Client die Verbindung anhand 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 WebSphere MQ Administrator kann eine offene Sitzung zwangsweise schließen und alle Sitzungsdaten löschen. 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 die Standardeinstellung `MqttConnectOptions` verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor die Clientverbindung hergestellt wird, werden bei der Herstellung der Clientverbindung alle alten Subskriptionen für den Client gelöscht. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` auf `false` setzen, bevor eine Verbindung hergestellt wird, werden alle Subskriptionen, die der Client erstellt, zu den Subskriptionen hinzugefügt, die bereits vor Herstellung der Verbindung für den Client existierten. 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 Veröffentlich-

ungen 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 `cleanSession`-Modus vor Herstellung der Verbindung festlegen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie den Modus von `cleanSession=false` in `cleanSession=true` ändern, werden alle bisherigen Subskriptionen für den Client gelöscht und alle noch nicht empfangenen Veröffentlichungen verworfen.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Client-ID

Die Client-ID ist eine aus 23 Byte bestehende Zeichenfolge, mit der ein MQTT-Client identifiziert wird. 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

bestimmte Vorgehensweise zur Zuordnung von Client-IDs zu nutzen. Dies ermöglicht die Konfiguration eines Clients mit der zugehörigen gewählten ID.

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? Müssen Sie eine Datenbank zu Rate ziehen, die jedes Gerät einer Client-ID und einem Server zuordnet? Gibt der Name des Geräts Aufschluss darüber, mit welchem Server es verbunden ist? In der Anzeige der MQTT-Clientverbindungen sehen Sie zu jeder Verbindung die Client-ID. Müssen Sie in einer Tabelle nachschlagen, um eine Client-ID einer physische Einheit 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? Wird von Ihnen eine neue ID angelegt? Wenn Sie ein physisches Gerät ändern, aber dieselbe ID beibehalten, werden ausstehende Veröffentlichungen und aktive Subskriptionen automatisch auf das neue Gerät übertragen.

Wie stellen Sie sicher, dass die Client-IDs eindeutig sind? 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 - beispielsweise unter Verwendung seiner MAC-Adresse? Oder verfügen Sie über einen Softwareinstallations- und Konfigurationsprozess, mit dem das Gerät vor seiner Aktivierung konfiguriert wird?

Sie können beispielsweise eine Client-ID 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
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere 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
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“.

Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Veröffentlichung "Last Will and Testament"

Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. Sie können dazu die Inhalte der Veröffentlichung und das Thema, an das sie gesendet werden sollen, vordefinieren. Bei "Last Will and Testament" handelt es sich um eine Verbindungseigenschaft. Legen Sie die Eigenschaft fest, bevor Sie eine Clientverbindung herstellen.

Erstellen Sie ein Thema für "Last Will and Testament". Sie können ein Thema wie `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`erstellen.

Richten Sie mit der Methode `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` ein "Last Will and Testament" ein.

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 WebSphere MQ persistent ist und die Zustellung garantiert werden kann. 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
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den

Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Nachrichtenpersistenz in MQTT-Clients

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Nachrichtenpersistenz in MQTT-Clients

Veröffentlichungsnachrichten werden zu persistenten Nachrichten, wenn sie mit der Servicequalität "Mindestens einmal" oder "Genau 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 WebSphere 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 "höchstens einmal" (`QoS=0`) angeben, löscht der Client die Nachricht, sobald sie veröffentlicht wurde. 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 WebSphere 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 vom Client entfernt, wenn sie nicht mehr benötigt wird, um sicherzustellen, dass sie mindestens einmal ("at least once"; `QoS=1`) oder genau einmal ("exactly once"; `QoS=2`) zugestellt wird.

2. Wenn eine Nachricht als QoS 1 oder 2 markiert ist, wird sie als persistente Nachricht in IBM MessageSight und IBM WebSphere MQ eingereiht. Wenn sie als QoS=0 markiert ist, wird sie als nicht persistente Nachricht in IBM MessageSight und IBM WebSphere MQ eingereiht. In IBM WebSphere MQ werden nicht persistente Nachrichten "genau einmal" zwischen Warteschlangenmanagern übertragen, es sei denn, im Nachrichtenkanal ist das Attribut NPMSPEED auf FASTgesetzt.

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 in Java SE ausgeführt wird, verfügt über eine Standardimplementierung der Schnittstellen, die persistente Nachrichten im Dateisystem speichern. Dabei wird das Paket `java.io` verwendet. Der Client hat auch eine Standardimplementierung für Java ME, `MqttDefaultMIDPPersistence`.

Persistenzklassen

`MqttClientPersistence`
Übergibt eine Instanz Ihrer Implementierung von `MqttClientPersistence` an den MQTT-Client als Parameter des `MqttClient`-Konstruktors. Wenn Sie den Parameter `MqttClientPersistence` vom `MqttClient`-Konstruktor übergehen, speichert der MQTT-Client persistente Nachrichten mithilfe der Klasse `MqttDefaultFilePersistence` oder `MqttDefaultMIDPPersistence`.

`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 Klassen `MqttDefaultFilePersistence` oder `MqttDefaultMIDPPersistence` 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 in einem Standardverzeichnis speichern. Der Name des Verzeichnisses lautet *`client identifier`*-tcp *`hostname portnumber`*. `"\", "\\", "/", ":"` und `" "` werden aus der Verzeichnisnamenszeichenfolge entfernt.

Der Pfad zum Verzeichnis entspricht dem Wert der Systemeigenschaft `rcp.data`. Wenn `rcp.data` nicht festgelegt ist, entspricht der Pfad dem Wert der Systemeigenschaft `usr.data`.

`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, der die Anwendung gestartet hat, gestartet wurde.

`MqttDefaultMIDPPersistence`
`MqttDefaultMIDPPersistence` enthält einen Standardkonstruktor und keine Parameter. Zum Speichern von Nachrichten wird das Paket `javax.microedition.rms.RecordStore` verwendet.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Die Nutzdaten von `MqttMessage` umfassen eine Bytefeldgruppe. Nachrichten sollten so klein wie möglich sein. Das MQTT-Protokoll ermöglicht 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 WebSphere 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 `clientId` und `qos`, aber dieser Inhalt kann sich in 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 kann der MQTT -Client ein temporäres Nachrichtenobjekt aus den Parametern der Methode `MqttTopic.publish(byte [] payload, int qos, boolean retained)` erstellen, wenn er eine Veröffentlichung erstellt.

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
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“.

Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Von einem MQTT-Client bereitgestellte Servicequalität

Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere 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.

Das MQTT-Protokoll verlangt nicht von Servern, Veröffentlichungen mit der Servicequalität `QoS=0` an einen Client weiterzuleiten. 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 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 durchführen kann ("PUBREL"). Wenn der Absender keine Bestätigung der Nachricht "PUBREL" empfängt, wird die Nachricht "PUBREL" solange erneut gesendet, bis eine Bestätigung empfangen wird. Der Absender löscht die bei ihm 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

Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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

Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"

Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen

Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere MQ identisch.

Ständige Veröffentlichungen und MQTT-Clients

Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema sofort an Sie weitergeleitet.

Geben Sie mithilfe der Methode `MqttMessage.setRetained` an, ob eine Veröffentlichung für ein Thema aufbewahrt wird oder nicht.

Zum Löschen einer ständigen Veröffentlichung in IBM WebSphere MQführen Sie den MQSC-Befehl **CLEAR TOPICSTR** aus.

Wenn Sie eine Veröffentlichung ohne Nutzdaten erstellen, wird die leere Veröffentlichung an Subskribenten weitergeleitet. Andere MQTT-Broker leiten eine leere Veröffentlichung möglicherweise nicht an Subskribenten weiter.

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.

Wenn Sie eine ständige Veröffentlichung erstellen oder aktualisieren, senden Sie die Veröffentlichung mit QoS oder 1 oder 2. Wenn Sie es mit der QoS 0 senden, erstellt IBM WebSphere MQ eine nicht persistente ständige Veröffentlichung. Die Veröffentlichung wird nicht beibehalten, wenn der Warteschlangenmanager gestoppt wird.

Verwenden Sie ständige Veröffentlichungen, um den neuesten Wert einer Messung aufzuzeichnen. Neue Subskribenten des Themas mit der ständigen Veröffentlichung empfangen 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.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen

ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere 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 WebSphere 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 WebSphere MQ-Befehl kann eine Subskription entfernen. Listen Sie Subskriptionen mit WebSphere MQ Explorer oder mit **runmqsc** -oder PCF-Befehlen auf. Alle MQTT-Clientsubskriptionen haben einen Namen. Sie erhalten einen Namen im folgenden Format: *`ClientIdentifier`*`:`*`Topic name`*

Wenn Sie die Standardeinstellung `MqttConnectOptions` verwenden oder `MqttConnectOptions.cleanSession` auf `true` setzen, bevor die Clientverbindung hergestellt wird, werden bei der Herstellung der Clientverbindung alle alten Subskriptionen für den Client gelöscht. Alle neuen Subskriptionen, die der Client während der Sitzung einrichtet, werden bei der Trennung der Clientverbindung entfernt.

Wenn Sie `MqttConnectOptions.cleanSession` auf `false` setzen, bevor eine Verbindung hergestellt wird, werden alle Subskriptionen, die der Client erstellt, zu den Subskriptionen hinzugefügt, die bereits vor Herstellung der Verbindung für den Client existierten. 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 `cleanSession`-Modus vor Herstellung der Verbindung festlegen; der Modus gilt für die gesamte Sitzung. Um den Modus zu ändern, müssen Sie die Clientverbindung trennen und wiederherstellen. Wenn Sie den Modus von `cleanSession=false` in `cleanSession=true` ändern, werden alle bisherigen Subskriptionen für den Client gelöscht und alle noch nicht empfangenen Veröffentlichungen 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
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"

Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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 WebSphere 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 WebSphere MQ identisch.

Mit Themenzeichenfolgen werden Veröffentlichungen an Subskribenten gesendet. Erstellen Sie mit der Methode `MqttClient.getTopic(java.lang.String topicString)` eine Themenzeichenfolge.

Mit Themenfiltern werden Themen subskribiert und Veröffentlichungen empfangen. Themenfilter können Platzhalterzeichen enthalten. Mithilfe von Platzhalterzeichen können mehrere Themen subskribiert werden. Erstellen Sie einen Topicfilter mit einer Subskriptionsmethode, z. B. `MqttClient.subscribe(java.lang.String topicFilter)`.

Themenzeichenfolgen

Die Syntax einer IBM WebSphere MQ-Themenzeichenfolge wird 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. Themenzeichenfolgen, die von MQTT -Clients an IBM WebSphere MQ gesendet werden, müssen der Konvention für Warteschlangenmanagernamen entsprechen. Vor allem dürfen Themenzeichenfolgen keine Silbentrennungsstriche enthalten.
2. Die maximale Länge ist unterschiedlich. IBM WebSphere 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 WebSphere Message Broker war ein Thema der Ebene null (`'...//...'`) ungültig. IBM WebSphere MQ unterstützt Themen der Ebene null.

Anders als die Publish/Subscribe-Funktion von IBM WebSphere 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 Ver-

waltungsthema in IBM WebSphere 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 WebSphere MQ -Themenfilters wird unter Themenbasiertes Platzhalterschemabeschrieben. Die Syntax der Themenfilter, die Sie mit einem MQTT-Client erstellen können, ist in der Klasse `MqttClient` in der API-Dokumentation für den MQTT-Client für Java beschrieben. Links zur Client-API-Dokumentation für die MQTT-Clientbibliotheken finden Sie unter MQTT client programming reference.

Die Syntax ist bei allen Themenfiltern nahezu identisch. Lediglich die Art und Weise, wie verschiedene MQTT-Broker einen Themenfilter interpretieren, ist unterschiedlich. In WebSphere Message Broker Version 6 kann ein Platzhalterzeichen für mehrere Ebenen nur am Ende eines Themenfilters verwendet werden. In IBM WebSphere MQ kann ein Platzhalterzeichen für mehrere Ebenen in jeder Ebene der Themenstruktur verwendet werden. Beispiel: `USA/#/Dutchess County`.

Zugehörige Konzepte

Callbacks und Synchronisierung in MQTT-Clientanwendungen
Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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
Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

Client-ID

Zustellungstoken

Veröffentlichung "Last Will and Testament"
Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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

Veröffentlichungen
Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT-Client kann Veröffentlichungen erstellen, die an IBM WebSphere MQ versendet werden sollen, und Themen in IBM WebSphere MQ MQ subskribieren, um Veröffentlichungen zu empfangen.

Von einem MQTT-Client bereitgestellte Servicequalität
Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal“, „Mindestens einmal“ und „Genau einmal“. Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

Ständige Veröffentlichungen und MQTT-Clients
Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema 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.

Konzepte zur Programmierung von C-Clients

In diesem Abschnitt werden Unterschiede zwischen dem C- und dem Java-Client für Version 3.1 von MQ Telemetry Transport beschrieben. Der Abschnitt ergänzt die Clientkonzepte und die C-Referenzinformationen.

Dieser Abschnitt ist auf die gleiche Weise organisiert wie „Konzepte zur Programmierung von MQTT-Clients" auf Seite 556. Jede Überschrift entspricht einem Thema in den Informationen zu den *Programmierungskonzepten für den WebSphere(r) MQ Telemetry Transport-Client*. In den Abschnitten werden die Unterschiede zwischen dem C-Client und dem Java-Client beschrieben. Geringfügige Unterschiede in den Signaturen der Java-Methoden und C-Funktionen werden nicht beschrieben.

Der C-Client wird am häufigsten dazu verwendet, einen einfachen Adapter zwischen einer Telemetrieeinheit und dem WebSphere MQ Telemetry-Dämon für Einheiten zu implementieren. Der Dämon wird häufig als Netzkonzentrator zwischen sehr einfachen Telemetriegeräten und dem Telemetrieservice (MQXR) verwendet.

Beim WebSphere MQ Telemetry-Dämon für Einheiten handelt es sich ebenfalls um einen C-Client und es werden die Abweichungen im Verhalten vom Telemetrieservice (MQXR) beschrieben. Der Dämon stellt keine Implementierung von JAAS oder SSL für Clients bereit, die eine Verbindung dazu herstellen.

`mqttclient.dll` und `mqttclient.lib` sind die 32-Bit-Windows-Bibliotheken, die Clientfunktionen für die C-Implementierung des Protokolls von MQ Telemetry Transport Version 3.1 enthalten. Die 32 -Bit- Linux -Bibliotheken sind `libmqttclient.so` und `libmqttclient.a`. Die folgenden beiden Headerdateien enthalten die Funktion und weitere Deklarationen, die für Clientanwendungen erforderlich sind: `MQTTClient.h` und `MQTTClientPersistence.h`. Diesen Dateien werden mit der Installation von WebSphere MQ Telemetry bereitgestellt.

Zum Entwickeln und Ausführen eines MQ Telemetry Transport-Clients müssen Sie diese Dateien auf die Clienteinheit kopieren. Anders als bei WebSphere MQ-Clients müssen Sie keine separate Clientlaufzeitkomponente installieren.

Lesen Sie die Lizenzbedingungen, die der WebSphere MQ Telemetry-Funktion zur Steuerung der Verbindung von WebSphere MQ Telemetry Transport-Clients mit WebSphere MQ und dem WebSphere MQ Telemetry-Dämon für Einheiten zugeordnet ist.

Beim C-Client handelt es sich um eine Referenzimplementierung von Version 3.1 von MQ Telemetry Transport. Sie können Ihre eigenen Clients in unterschiedlichen Sprachen implementieren, die für unterschiedliche Einheitenplattformen geeignet sind. Ausführliche Informationen hierzu finden Sie unter MQ Telemetry Transport-Format und -Protokoll.

MQTT-Client-ID

„Client-ID" auf Seite 561	Die Client-ID ist eine aus 23 Byte bestehende Zeichenfolge, mit der ein MQTT-Client identifiziert wird. 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 bestimmte Vorgehensweise zur Zuordnung von Client-IDs zu nutzen. Dies ermöglicht die Konfiguration eines Clients mit der zugehörigen gewählten ID.

- Keine Unterschiede

Veröffentlichungen

„Veröffentlichungen” auf Seite 569	Veröffentlichungen sind Instanzen von `MqttMessage` , die einer Themenzeichenfolge zugeordnet sind. MQTT

- Die Callback-Funktion wird für Veröffentlichungen mit der Servicequalität "Fire and Forget" (`QoS=0`) nicht aufgerufen.

Zustellungstoken

„Zustellungstoken” auf Seite 563	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.

- Ein Zustellungstoken hat den Wert `int`. Die Angabe für typedef lautet `MQTTClient_deliveryToken`.
- Die Callback-Funktion wird für Veröffentlichungen mit der Servicequalität "Fire and Forget" (`QoS=0`) nicht aufgerufen.

Ständige Veröffentlichungen

„Ständige Veröffentlichungen und MQTT-Clients” auf Seite 573	Wenn Sie eine Subskription für ein Thema erstellen, zu dem es eine ständige Veröffentlichung gibt, wird die neueste ständige Veröffentlichung zu dem Thema sofort an Sie weitergeleitet.

- Nachrichten einer ständigen Veröffentlichung werden im Dämon nur gespeichert, wenn die Persistenz konfiguriert ist. Weitere Informationen hierzu finden Sie im Abschnitt zum Speichern von Nachrichten einer ständigen Veröffentlichung und Subskriptionen.

  In WebSphere MQ legt die Servicequalität fest, ob eine Nachricht einer ständigen Veröffentlichung dauerhaft gespeichert wird. Wenn dem Telemetrieservice ein Client angehängt ist, werden Nachrichten einer ständigen Veröffentlichung mit der Servicequalität "Fire and Forget" (`QoS=0`) gelöscht, sobald der Warteschlangenmanager beendet.

Subskriptionen

„Subskriptionen” auf Seite 574	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.

- Permanente Subskriptionen werden im Dämon nur gespeichert, wenn die Persistenz konfiguriert ist. Weitere Informationen hierzu finden Sie im Abschnitt zum Speichern von Nachrichten einer ständigen Veröffentlichung und Subskriptionen.
- Veröffentlichungen können synchron empfangen werden. Rufen Sie die Funktion `MQTTClient_receive` auf.

Callbacks und Synchronisation

„Callbacks und Synchronisierung in MQTT-Clientanwendungen" auf Seite 556	Das MQTT-Clientprogrammiermodell arbeitet sehr viel mit Threads. 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.

- Die Synchronisierung wird im C-Client modal ausgeführt. Beim Aufrufen von `MQTTClient_setCallback` wird der Client in den asynchronen Modus versetzt.
- Im synchronen Modus muss der Anwendungsclient aus freien Stücken die Steuerung abgeben, damit der MQTT-Client Bestätigungen verarbeiten und MQTT-Pings absetzen kann, um die Funktionen des Netzes aufrechtzuerhalten. Geben Sie die Steuerung durch Aufrufen von `MQTTClient_receive` oder `MQTTClient_yield` ab.

Themenzeichenfolgen und Filter

„Themenzeichenfolgen und Themenfilter in MQTT-Clients" auf Seite 576	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 WebSphere MQ identisch.

- Der WebSphere MQ Telemetry-Dämon für Einheiten bearbeitet das Platzhalterzeichen für mehrere Ebenen (*#*) anders als WebSphere MQ Version 7. */#* müssen die letzten beiden Zeichen der Filterzeichenfolge sein, damit *#* als Platzhalterzeichen funktionieren kann. In WebSphere MQ v7ist `../#/..` eine gültige Verwendung des Platzhalterzeichens für mehrere Ebenen. Der WebSphere MQ Telemetry-Dämon für Einheiten verarbeitet das Platzhalterzeichen für mehrere Ebenen auf die gleiche Weise wie WebSphere MQ Broker Version 6.

Servicequalität

„Von einem MQTT-Client bereitgestellte Servicequalität" auf Seite 571	Ein MQTT-Client stellt drei Servicequalitätsstufen für die Zustellung von Veröffentlichungen an WebSphere MQ und an den MQTT-Client bereit: „Höchstens einmal", „Mindestens einmal" und „Genau einmal". Wenn ein MQTT-Client eine Anforderung zum Erstellen einer Subskription an WebSphere MQ sendet, wird die Anforderung mit der Servicequalität "at least once" (Mindestens einmal) gesendet.

- Keine Unterschiede

Nachrichtenpersistenz

„Nachrichtenpersistenz in MQTT-Clients" auf Seite 566	Veröffentlichungsnachrichten werden zu persistenten Nachrichten, wenn sie mit der Servicequalität "Mindestens einmal" oder "Genau 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.

- Legen Sie den Mechanismus für die Nachrichtenpersistenz aufgrund der Unterschiede in der programmiersprachenbezogenen Bindung im C-Client folgendermaßen fest. Rufen Sie den MQTT-C-Client mit einer der drei Optionen auf, die als vierter Parameter für `MQTTClient_create` festgelegt sind:

  `MQTTCLIENT_PERSISTENCE_DEFAULT`
  Dabeibasierte Persistenz; die zugehörigen Einzelheiten sind spezifisch für die Clientplattform.

MQTTCLIENT_PERSISTENCE_NONE
Daten werden nur im Hauptspeicher gespeichert und gehen verloren, wenn der Client beendet wird. Der WebSphere MQ Telemetry-Dämon für Einheiten unterstützt nur diese Option.

MQTTCLIENT_PERSISTENCE_USER
Die können Funktionen zur Implementierung Ihrer eigenen Persistenzmechanismen entwickeln. Übergeben Sie eine Struktur (`MQTTClient_persistence`) mit Verweisen auf Ihre Funktionen an den Aufruf `MQTTClient_create`. Einzelheiten finden Sie in den Referenzinformationen zum MQTT-C-Client.

Sitzungen bereinigen

„Sitzungen bereinigen” auf Seite 560	Der MQTT-Client und der Telemetrieservice (MQXR) verwalten Sitzungsstatusinformationen. Mithilfe der Statusinformationen werden „Mindestens einmal“- und „Genau einmal“-Zustellungen und der „Genau einmal“-Empfang von Veröffentlichungen sichergestellt. Der Sitzungsstatus schließt auch Subskriptionen ein, die von einem MQTT-Client erstellt wurden. Sie können festlegen, ob ein MQTT-Client mit oder ohne Verwaltung von Statusinformationen zwischen Sitzungen ausgeführt werden soll. Ändern Sie den Bereinigungssitzungsmodus, indem Sie `MqttConnectOptions.cleanSession` vor dem Herstellen der Verbindung festlegen.

- Keine Unterschiede

Last Will and Testament

„Veröffentlichung "Last Will and Testament"” auf Seite 565	Für den Fall, dass eine MQTT-Clientverbindung unerwartet beendet wird, können Sie WebSphere MQ Telemetry so konfigurieren, dass es eine Veröffentlichung "Letzter Wille und Testament" sendet. 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.

- Keine Unterschiede

Programmfehler behandeln

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:

- Ursachencodes für alle anderen WebSphere MQ-Plattformen

Gestalten Sie Ihre Programme so, dass alle Rückgabecodes bearbeitet werden, die ein Aufruf ausgeben kann.

Systemunterbrechungen

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:

UNIX-, Linux- und Windows-Systeme
In diesen Umgebungen können Sie Ihre MQPUT- und MQGET-Aufrufe wie gewohnt ausgeben, Sie müssen allerdings Synchronisationspunkte mithilfe der MQCMIT- und MQBACK-Aufrufe ausgeben (siehe „Arbeitseinheiten per Commit festschreiben und per Backout zurücksetzen” auf Seite 343). 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. Bei WebSphere MQ auf UNIX-, Linux-und Windows -Systemen schlägt ein MQGET-oder MQPUT-Aufruf in Ihrer Anwendung fehl, wenn alle Protokolldateien mit der Nachricht MQRC_RESOURCE_PROBLEM gefüllt werden. Weitere Informationen zu Protokolldateien auf AIX-, HP-UX-, Linux-, Solaris-und Windows -Systemen finden Sie unter Verwaltung .

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 UNIX-, Linux- und Windows-Systemen 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.

Auf Systemen unter WebSphere MQ for Windows, UNIX und Linux wird der Rücksetzungszähler auch nach dem Neustart des Warteschlangenmanagers fortgesetzt.

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 12.

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 422.

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 12.

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, fügt er der Nachricht einen Header hinzu, dessen Format durch die Struktur des Headers für nicht zustellbare Nachrichten (MQDLH) beschrieben wird; siehe MQDLH-Header für nicht zustellbare Nachrichten . 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 empfiehlt sich, allen Nachrichten, die in die Warteschlange für nicht zustellbare Nachrichten eingereiht werden, eine MQDLH-Struktur hinzuzufügen. Soll die von manchen WebSphere MQ-Produkten bereitgestellte Steuerroutine für nicht zustellbare Nachrichten verwendet werden, ***muss*** den Nachrichten sogar unbedingt eine MQDLH-Struktur hinzugefügt werden.

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 585.

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 unter WebSphere MQ for z/OS mit CICS gearbeitet wird und ein Nachrichtenkanalagent diese Nachricht in die Warteschlange für nicht zustellbare Nachrichten einreiht, wird das Feld *PutApplType*

auf MQAT_CICS gesetzt und das Feld *PutApplName* enthält die Anwendungs-ID *ApplId* 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

Hier finden Sie Informationen zu den WebSphere MQ Multicast-Programmierungstasks wie z. B. zum Herstellen einer Verbindung zu einem Warteschlangenmanager und zum Melden von Ausnahmebedingungen.

WebSphere 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 WebSphere MQ-Anwendungen kein wesentliches Umschreiben erforderlich, damit Multicast verwendet werden kann. Es müssen jedoch möglicherweise einige Einschränkungen (weitere Informationen dazu finden Sie im Abschnitt „Multicasting und das Message Queue Interface” auf Seite 587) und Sicherheitsprobleme (siehe Multicast-Sicherheit) beachtet werden.

Multicasting und das Message Queue Interface

In diesem Abschnitt finden Sie Informationen, die Ihnen das Verständnis der wichtigsten MQI-Konzepte und deren Beziehung zu WebSphere 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.

WebSphere 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 23.

In der folgenden Tabelle werden alle wichtigen MQI-Konzepte und deren Beziehung zu Multicast aufgeführt:

Tabelle 71. 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

Tabelle 71. MQI-Konzepte und deren Beziehung zu Multicast (Forts.)

MQI-Konzept	Aktion beim Versuch, Multicast zu verwenden	Ursachencode
MQINQ	Abgelehnt für Themenkennungen: MQINQ- und MQSET-Aufrufe von Themen werden nicht unterstützt.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
	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
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

Tabelle 71. MQI-Konzepte und deren Beziehung zu Multicast (Forts.)		
MQI-Konzept	**Aktion beim Versuch, Multicast zu verwenden**	**Ursachencode**
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 WebSphere MQ Multicast vertraut zu machen. Obwohl WebSphere MQ eine Verbindung zu einem WebSphere 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 zur automatischen Verbindungswiederholung für den Client.

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 in Überwachung von Multicastanwendungen.

Datenkonvertierung in der MQI für die Multicast-Nachrichtenübertragung

In diesen Informationen erfahren Sie, wie die Datenkonvertierung für die WebSphere MQ Multicast-Nachrichtenübertragung funktioniert.

Bei WebSphere MQ Multicast handelt es sich um ein gemeinsam genutztes verbindungsunabhängiges Protokoll und daher kann nicht jeder Client spezielle Anforderungen zur Datenkonvertierung stellen. Jeder Client, der den gleichen Multicastdatenstrom subskribiert, empfängt die gleichen Binärdaten; wenn eine Konvertierung von WebSphere MQ-Daten erforderlich ist, wird die Konvertierung deshalb lokal auf jedem Client ausgeführt.

Die Daten werden auf dem Client für den WebSphere MQ Multicast-Datenverkehr 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 442 finden Sie Informationen darüber, welche Bibliotheken sich jetzt in den Client- und Serverpaketen befinden.

Informationen zur Verwaltung der Datenkonvertierung finden Sie unter Datenkonvertierung für die Multicast-Nachrichtenübertragung aktivieren.

Weitere Informationen zur Datenkonvertierung finden Sie unter Datenkonvertierung .

Weitere Informationen zu Datenkonvertierungsexits und `ClientExitPath`finden Sie im Abschnitt Zeilengruppe`ClientExit`der Clientkonfigurationsdatei.

Erstellen von Ausnahmeberichten für Multicast

Hier finden Sie Informationen zu WebSphere MQ Multicast-Ereignishandlern und zur Berichterstellung über WebSphere MQ Multicast-Ausnahmen.

WebSphere MQ Multicast unterstützt Sie bei der Fehlerbestimmung, indem der Ereignishandler aufgerufen wird, um mithilfe des standardmäßigen WebSphere MQ-Ereignishandler-Mechanismus Multicastereignisse zu dokumentieren.

Ein einzelnes Multicastereignis kann dazu führen, dass mehrere WebSphere MQ-Ereignisse aufgerufen werden, da möglicherweise mehrere `MQHCONN`-Verbindungskennungen den gleichen Multicastsender oder -empfänger verwenden. Für jede Multicastausnahme wird jedoch pro WebSphere MQ-Verbindung nur ein Ereignishandler aufgerufen.

Die WebSphere MQ-Konstante `MQCBDO_EVENT_CALL` ermöglicht es Anwendungen, einen Callback zu registrieren, damit nur WebSphere MQ-Ereignisse empfangen werden; die Konstante `MQCBDO_MC_EVENT_CALL` ermöglicht es Anwendungen, einen Callback zu registrieren, damit nur Multicastereignisse empfangen werden. Wenn beide Konstanten verwendet werden, werden beide Ereignistypen empfangen.

Multicast-Ereignisse anfordern

WebSphere 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, wird der Ereignishandler nur WebSphere MQ Multicast-Ereignisse anstelle von Ereignissen auf Verbindungsebene 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

Die WebSphere MQ Multicast-Ausnahmen enthalten unterstützende Informationen, die im Parameter **`Buffer`** der Callback-Funktion angegeben 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 72. 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
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

Tabelle 72. Beschreibung des Multicast-Ereigniscodes (Forts.)

Ereigniscode	Beschreibung	Zusätzliche Daten
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	
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

Tabelle 72. Beschreibung des Multicast-Ereigniscodes (Forts.)		
Ereigniscode	**Beschreibung**	**Zusätzliche Daten**
MQMCEV_PORT_INTERFACE_FAILOVER	Die von einem Empfänger-rmmPort verwendete Netzschnittstelle ist ausgefallen; die Überbrückung zu einer anderen Schnittstelle wurde erfolgreich abgeschlossen	RMM-Konfiguration

.NET verwenden

WebSphere MQ-Klassen für .NET ermöglichen es einem Programm, das im .NET-Programmierungsframework geschrieben wurde, eine Verbindung zu WebSphere MQ als WebSphere MQ-MQI-Client herzustellen oder sich direkt mit einem WebSphere MQ-Server zu verbinden.

Wenn Sie über Anwendungen verfügen, die mit Microsoft .NET Framework arbeiten, und die Funktionen von WebSphere MQ nutzen möchten, müssen Sie WebSphere MQ-Klassen für .NET verwenden.

Die objektorientierte WebSphere MQ .NET-Schnittstelle unterscheidet sich von der MQI-Schnittstelle dahingehend, dass sie Methoden von Objekten statt MQI-Verben verwendet.

Die prozedurale WebSphere MQ-Anwendungsprogrammierschnittstelle basiert auf Verben. In der nachfolgenden Liste sind einige dieser Verben aufgeführt:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Diese Verben verwenden alle als Parameter eine Kennung für das WebSphere MQ-Objekt, auf dem sie ausgeführt werden sollen. Da .NET objektorientiert arbeitet, ist dieser Sachverhalt für die .NET-Programmierschnittstelle umgekehrt. Ihr Programm besteht aus einer Gruppe von WebSphere MQ-Objekten, mit denen Sie arbeiten, indem Sie Methoden zu diesen Objekten aufrufen. Sie können Programme in jeder von .NET unterstützten Sprache schreiben.

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 WebSphere MQ-Klassen für .NET handelt es sich um eine Gruppe von Klassen, über die .NET-Anwendungen mit WebSphere MQ interagieren können. Sie stellen die verschiedenen Komponenten von WebSphere MQ dar, die Ihre Anwendung verwendet. Hierzu gehören z. B. Warteschlangenmanager, Warteschlangen, Kanäle und Nachrichten. Detaillierte Informationen über diese Klassen finden Sie im Abschnitt Klassen und Schnittstellen von WebSphere MQ .NET.

Bevor Sie Anwendungen kompilieren können, die Sie schreiben, müssen Sie .NET Framework auf Ihrem System installieren. Anweisungen zur Installation von WebSphere MQ-Klassen für .NET und .NET Framework finden Sie in „WebSphere MQ-Klassen für .NET installieren” auf Seite 596.

Zugehörige Konzepte

Technische Übersicht

„Verbindungsoptionen” auf Seite 595
Es gibt drei Möglichkeiten, WebSphere MQ-Klassen für .NET mit einem Warteschlangenmanager zu verbinden. Prüfen Sie, welcher Verbindungstyp für Ihre Anforderungen am besten geeignet ist.

„WebSphere MQ-Klassen für .NET verwenden” auf Seite 607
In den nachfolgend genannten Abschnitten wird beschrieben, wie Ihr System für die Ausführung der Musterprogramme zur Überprüfung Ihrer WebSphere MQ-Klassen für .NET-Installation konfiguriert wird und wie Sie eigene Programme ausführen.

„WebSphere MQ .NET-Probleme beheben” auf Seite 610
Wenn ein Programm nicht erfolgreich beendet wird, führen Sie eine der Beispielanwendungen aus und halten Sie sich an die Ratschläge in den Diagnosenachrichten.

„.NET-Programme von WebSphere MQ schreiben und implementieren” auf Seite 611
Wenn Sie WebSphere MQ-Klassen für .NET nutzen möchten, um auf WebSphere MQ-Warteschlangen zuzugreifen, schreiben Sie Programme in einer von .NET unterstützten Sprache, die Aufrufe enthält, die Nachrichten in WebSphere MQ-Warteschlangen einreihen und daraus abrufen.

„Angepasster IBM WebSphere MQ-Kanal für Microsoft Windows Communication Foundation (WCF)” auf Seite 631
Der angepasste WCF-Kanal (WCF = Microsoft Windows Communication Foundation) für IBM WebSphere MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„Anwendungen entwickeln” auf Seite 7
IBM WebSphere MQ bietet mehrere Möglichkeiten, mit denen Sie Anwendungen entwickeln können, um Nachrichten zur Unterstützung Ihrer Geschäftsprozesse zu senden und zu empfangen. Darüber hinaus können Sie Anwendungen für das Management Ihrer Warteschlangenmanager und zugehörigen Ressourcen entwickeln.

WebSphere MQ-Klassen für .NET - Erste Schritte

WebSphere MQ-Klassen für .NET ermöglichen es einem Programm, das im .NET-Programmierungsframework geschrieben wurde, eine Verbindung zu WebSphere MQ als WebSphere MQ-MQI-Client herzustellen oder sich direkt mit einem WebSphere MQ-Server zu verbinden.

Verbindungsoptionen

Es gibt drei Möglichkeiten, WebSphere MQ-Klassen für .NET mit einem Warteschlangenmanager zu verbinden. Prüfen Sie, welcher Verbindungstyp für Ihre Anforderungen am besten geeignet ist.

Verbindungen über Clientbindungen

Wenn Sie die WebSphere MQ-Klassen für .NET als WebSphere MQ-MQI-Client verwenden möchten, können Sie sie mit dem WebSphere MQ-MQI-Client auf der WebSphere MQ-Servermaschine oder auf einem separaten System installieren. Eine Clientverbindung kann XA- oder Nicht-XA-Transaktionen verwenden.

Verbindungen über Serverbindungen

Bei Verwendung des Serverbindungsmodus wird die Kommunikation der WebSphere MQ-Klassen für .NET nicht über das Netz, sondern über die API des Warteschlangenmanagers ausgeführt. Dadurch wird die Leistung für WebSphere MQ-Anwendungen im Vergleich zur Verwendung von Netzverbindungen verbessert.

Wenn Sie eine Verbindung über Bindungen verwenden möchten, müssen Sie auf dem WebSphere MQ-Server die WebSphere MQ-Klassen für .NET installieren.

Verwaltete Clientverbindung

Bei diesem Verbindungsmodus wird ein WebSphere MQ-Client mit einem WebSphere MQ-Server verbunden, der auf einem lokalen oder fernen System ausgeführt wird.

Die in diesem Modus verbundenen WebSphere MQ-Klassen für .NET verbleiben in von .NET verwaltetem Code und rufen keine nativen Services auf. Weitere Informationen zu verwaltetem Code finden Sie in der Microsoft-Dokumentation.

Die Nutzung des verwalteten Clients unterliegt einer Reihe von Einschränkungen. Weitere Informationen hierzu enthält „Verwaltete Clientverbindungen” auf Seite 611.

WebSphere MQ-Klassen für .NET installieren

WebSphere MQ-Klassen für .NET wird (einschließlich Beispielkomponenten) zusammen mit WebSphere MQ installiert. Voraussetzung für die Installation ist Microsoft .NET Framework.

Die aktuellste Version von WebSphere MQ-Klassen für .NET wird standardmäßig als Teil der WebSphere MQ-Standardinstallation in der Funktion *Java and .NET Messaging and Web Services* installiert. Installationsanweisungen finden Sie unter IBM WebSphere MQ -Server auf Windows oder IBM WebSphere MQ -Client auf Windows -Systemen installieren .

Wenn Sie die WebSphere MQ-Klassen für .NET zuvor in einer Umgebung mit mehreren Installationen als Support-Pack installiert haben, kann WebSphere MQ erst installiert werden, wenn das Support-Pack deinstalliert wurde. Die mit WebSphere MQ installierte Funktion für WebSphere MQ-Klassen für .NET enthält die gleichen Funktionen wie das Support-Pack.

Es werden auch Beispielanwendungen bereitgestellt, einschließlich Quellendateien; siehe „Beispielanwendungen” auf Seite 608.

Zur Ausführung von WebSphere MQ-Klassen für .NET auf 32-Bit- oder 64-Bit-Plattformen muss Microsoft .NET Framework V2.0 oder höher installiert sein.

Anmerkung: Wenn Microsoft .NET Framework v2.0 oder höher nicht installiert wird, bevor WebSphere MQ V7.0.1 installiert wird, dann kann die WebSphere MQ-Produktinstallation weiterhin ohne Fehler verwendet werden, die WebSphere MQ-Klassen für .NET stehen dann jedoch nicht zur Verfügung. Wenn .NET Framework nach der Installation von WebSphere MQ 7.0.1installiert wird, müssen die WebSphere .NET-Assemblys mit dem Script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd registriert werden. Dabei ist *WMQInstallDir* das Verzeichnis, in dem WebSphere MQ 7.0.1 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 WebSphere MQ-Kanals für Microsoft WCF mit .NET 3 finden Sie im Abschnitt „Angepasster IBM WebSphere MQ-Kanal für Microsoft Windows Communication Foundation (WCF)” auf Seite 631.

Dezentrale Transaktionen 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 infor-

mieren. Wenn der Ressourcenmanager den Transaktionsmanager über seine Beteiligung informiert, erhält der Ressourcenmanager Callbacks vom Transaktionsmanager, wenn die Transaktion festgeschrieben oder rückgängig gemacht wird.

Die WebSphere MQ .NET-Klassen unterstützen bereits die dezentrale Transaktionsverarbeitung für Verbindungen im nicht verwalteten Modus und im Serverbindungsmodus. In diesen Modi delegieren WebSphere MQ .NET-Klassen alle Aufrufe an den erweiterten Transaktionsclient für die Programmiersprache C, der die Transaktionsverarbeitung im Auftrag von .NET verwaltet.

Die WebSphere MQ .NET-Klassen unterstützen jetzt die dezentrale Transaktionsverarbeitung im verwalteten Modus, in dem WebSphere MQ .NET-Klassen den Namensbereich 'System.Transactions' zur Unterstützung der dezentralen Transaktionsverarbeitung verwenden. Die System.Transactions-Infrastruktur ermöglicht eine einfache und effiziente transaktionsorientierte Programmierung, indem die Transaktionen unterstützt werden, die in allen Ressourcenmanagern einschließlich WebSphere MQ eingeleitet werden. Die WebSphere MQ .NET-Anwendung kann Nachrichten mithilfe des .NET-Modells für die implizite oder explizite Programmierung von 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.

WebSphere MQ .NET verwendet Microsoft Distributed Transaction Coordinator (MS DTC) als Transaktionsmanager. Dieser koordiniert und verwaltet die Transaktion zwischen mehreren Ressourcenmanagern. WebSphere MQ wird als Ressourcenmanager verwendet.

WebSphere MQ .NET orientiert sich am 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 598
- 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 Vorbereitungsaufruf abgesetzt wurde, gibt WebSphere MQ .NET die Vorbereitungsanforderung zurück, indem die Durchführung eines Rollbacks für die 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.

Verhaltensunterschiede zwischen verwalteter und nicht verwalteter Transaktionsunterstützung für Client mithilfe des Microsoft .NET-Namensbereichs 'System.Transactions'

Verschachtelte Transaktionen verfügen über einen Transaktionsbereich, der sich in einem anderen Transaktionsbereich befindet

- Der vollständig verwaltete Client für WebSphere MQ .NET unterstützt den verschachtelten Transaktionsbereich.
- Der nicht verwaltete Client für WebSphere MQ .NET unterstützt den verschachtelten Transaktionsbereich nicht.

Abhängige Transaktionen aus System.Transactions

- Der vollständig verwaltete Client für WebSphere MQ .NET unterstützt die Funktion für abhängige Transaktionen, die von 'System.Transactions' bereitgestellt wird.
- Der nicht verwaltete Client für WebSphere MQ .NET unterstützt die Funktion für abhängige Transaktionen nicht, die von 'System.Transactions' bereitgestellt wird.

Produktbeispiele

Neue 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 601

Dezentrale Transaktionsverarbeitung im verwalteten Modus

WebSphere 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 WebSphere MQ .NET-Klassen wird ein explizites Programmiermodell auf Basis der Klasse 'System.Transactions.Transaction' und ein implizites Programmiermodell auf Basis der Klasse 'System.Transactions.TransactionScope' bereitgestellt, in denen die Transaktionen automatisch von der Infrastruktur verwaltet werden.

Implizite Transaktion

Im folgenden Codeteil wird beschrieben, wie eine WebSphere MQ .NET-Anwendung eine Nachricht mithilfe der impliziten Programmierung von Transaktionen für .NET 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 WebSphere MQ .NET-Anwendung eine Nachricht mithilfe des .NET-Modells für die explizite Programmierung von Transaktionen 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

WebSphere MQ .NET-Klassen unterstützten nicht verwaltete Verbindungen (Client) über den erweiterten Transaktionsclient und COM+/MTS als Transaktionskoordinator. Dabei wird entweder das Modell für die implizite oder explizite Programmierung von Transaktionen verwendet. Im nicht verwalteten Modus delegieren WebSphere 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. WebSphere MQ .NET-Klassen stellen diese Unterstützung im Modus für den nicht verwalteten Transport (C-Client) bereit. Weitere Informationen finden Sie im Abschnitt XA-konforme Transaktionsmanager konfigurieren.

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 MTS installiert ist, wird Windows NT ein separater Service mit der Bezeichnung Microsoft Distributed Transaction Coordinator (MSDTC) hinzugefügt. Der MSDTC koordiniert die Transaktionen, die über mehrere separate Datenspeicher oder Ressourcen hinweg ausgeführt werden. Für die Ausführung muss in jedem Datenspeicher ein eigener proprietärer Ressourcenmanager implementiert werden.

WebSphere MQ wird durch die Implementierung einer Schnittstelle (proprietäre Ressourcenmanagerschnittstelle) mit MSDTC kompatibel. In dieser Schnittstelle werden DTC XA-Aufrufe den WebSphere MQ (X/Open)-Aufrufen zugeordnet. WebSphere MQ wird als Ressourcenmanager verwendet.

Wenn eine Komponente wie beispielsweise COM+ Zugriff auf WebSphere 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 WebSphere 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 WebSphere MQ ausgibt, um die Transaktion festzuschreiben oder rückgängig zu machen.

WebSphere MQ .NET-Anwendung mit nicht verwaltetem 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   UnmanagedToManaged
  Xa.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.EnterpriseServi
ces.TransactionOption.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 WebSphere 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 WebSphere 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 für eine Warteschlange `queue://queueName` und für ein Thema `topic://topicNamean`.

`-host`

Dies kann ein Hostname (z. B. localhost) oder eine IP-Adresse sein.

`-port`

Der Port, auf dem der Warteschlangenmanager ausgeführt wird.

`-channel`

Der Kanal, der für die Verbindung verwendet wird. Der Standardwert ist `SYSTEM.DEF.SVRCONN`.

`-transaction`

Das Ergebnis der Transaktion, z. B. Festschreiben oder Rollback.

`-mode`

Der Transportmodus, z. B. verwalteter oder nicht verwalteter Modus.

`-numberOfMsgs`

Die Anzahl der Nachrichten. Der Standardwert ist 1.

Beispiel

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Wiederherstellung von Transaktionen

In diesem Abschnitt wird der Prozess zum Wiederherstellen von Transaktionen in WebSphere 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 (Microsoft XA-Protokoll) basierenden Ressourcenmanager stellen die Optionen zum Konfigurieren des DTC-Dienstes 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 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 wiederherzustellen. Wenn die Verbindung erneut hergestellt werden kann, leitet der Transaktionsmanager die XA-Wiederherstellung ein.

Der Namensbereich 'System.Transactions' stellt die verwaltete Implementierung der dezentralen Transaktionsverarbeitung bereit, die auf dem MS-DTC als Transaktionsmanager basiert. Es werden ähnliche Funktionen wie die der nativen MS-DTC-Schnittstelle bereitgestellt, allerdings in einer vollständig verwalteten Umgebung. Der einzige Unterschied ist die Wiederherstellung der Transaktion. In 'System.Transactions' wird erwartet, dass Ressourcenmanager die Wiederherstellung selbst ausführen und anschließend mit den Transaktionsmanagern (MS-DTC) koordinieren. Ressourcenmanager müssen die Wiederherstellung einer bestimmten unvollständigen Transaktion anfordern. Dies wird anschließend vom Transaktionsmanager akzeptiert und auf Basis der tatsächlichen Ausgabe dieser bestimmten Transaktion koordiniert.

Transaktionswiederherstellungsprozess für WebSphere MQ .NET

In diesem Abschnitt wird beschrieben, wie verteilte Transaktionen mit WebSphere 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. WebSphere MQ .NET-Klassen verwenden eine ähnliche Vorgehensweise. Die Wiederherstellungsinformationen zur Transaktion werden in einer Systemwarteschlange mit der Bezeichnung SYSTEM.DOTNET.XARECOVERY.QUEUE protokolliert.

Die Transaktionswiederherstellung in WebSphere MQ .NET ist ein zweistufiger Prozess.

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 WebSphere MQ .NET XA-Überwachungsanwendung überwacht einen angegebenen Warteschlangenmanager und führt eine Wiederherstellung von unvollständigen Transaktionen aus, falls solche Transaktionen 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, der Warteschlangenmanager von WebSphere MQ jedoch ausgefallen ist.
- Wenn die Transaktion vorbereitet ist, aber der Transaktionsmanager inaktiv wird.

  Die Überwachungsanwendung muss über das gleiche System ausgeführt werden, auf dem Ihre WebSphere 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 Überwachungsanwen-

dung 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.

Anwendungsfälle für die Transaktionswiederherstellung für WebSphere MQ .NET

Im Folgenden sind die unterschiedlichen Einsatzszenarios aufgeführt:

- **WebSphere MQ-Anwendung mit einzelnem DTC-Dienst und einzelner Warteschlangenmanagerinstanz:** Wenn Sie in diesem Szenario 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 sie fehlschlägt und nicht vollständig beendet werden kann.

  In diesem Szenario wird eine einzelne Instanz der Überwachungsanwendung ausgeführt, da den Transaktionen ein einziger Warteschlangenmanager zugeordnet ist.

- **Mehrere WebSphere MQ-Anwendungen mit einzelnem DTC-Dienst und einzelner Warteschlangenmanagerinstanz:** In diesem Szenario sind mehrere WMQ-Anwendungen in einem einzelnen DTC-Dienst vorhanden und alle sind mit dem gleichen Warteschlangenmanager verbunden und führen Arbeitseinheiten in Transaktionen aus.

  Wenn die Transaktionen fehlschlagen und unvollständig sind, werden die Transaktionen, die alle Anwendungen betreffen, von der Überwachungsanwendung wiederhergestellt und abgeschlossen.

  In diesem Szenario wird eine einzelne Überwachungsanwendung ausgeführt, da in den Transaktionen ein Warteschlangenmanager verwendet wird.

- **Mehrere WebSphere MQ-Anwendungen, mehrere DTC-Dienste und unterschiedliche Warteschlangenmanagerinstanzen:** In diesem Szenario 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 Szenario 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 WebSphere MQ-Anwendungen, mehrere DTC-Dienste, mehrere Warteschlangenmanagerinstanzen:** In diesem Szenario sind mehrere WMQ-Anwendungen in verschiedenen DTC-Diensten vorhanden (d. h., jede Anwendung wird auf einem anderen System ausgeführt) und 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 Szenario wird pro Client (Benutzer oder Computer) nur eine einzelne Instanz der Überwachungsanwendung ausgeführt, da jeder Client über eine eigene Warteschlangenmanagerzuordnung verfügt, die in Transaktionen verwendet wird.

- **Mehrere WebSphere MQ-Anwendungen, einzelner DTC-Dienst, verschiedene Warteschlangenmanagerinstanzen:** In diesem Szenario 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 Szenario werden so viele Instanzen der Überwachungsanwendung ausgeführt, wie Verbindungen zu Warteschlangenmanagern 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 XA-Überwachungsanwendung 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 WebSphere MQ .NET-Klassen geschrieben wurden.

Befehl zum Starten der Überwachungsanwendung

```
WmqDotnetXAMonitor.exe -m <QueueManagerName> -n <ConnectionName> -c <Channel> -i
```

Dabei gilt:

- **n** - Verbindungsname im Host-Format (Port). 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** - Kanalname.
- **m** - Name des Warteschlangenmanagers Optional
- **i** -Heuristischer Verzweigungsabschluss. 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. Zum Lieferumfang von WebSphere MQ .NET gehört eine Beispieldatei für eine Anwendungskonfiguration. 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/Rollback
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/Rollback
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

WebSphere MQ-Klassen für .NET verwenden

In den nachfolgend genannten Abschnitten wird beschrieben, wie Ihr System für die Ausführung der Musterprogramme zur Überprüfung Ihrer WebSphere MQ-Klassen für .NET-Installation konfiguriert wird und wie Sie eigene Programme ausführen.

Warteschlangenmanager für das Akzeptieren von TCP/IP-Clientverbindungen konfigurieren

Gehen Sie folgendermaßen vor, um einen Warteschlangenmanager für eingehende Verbindungsanforderungen von Clients zu konfigurieren:

1. Definieren Sie einen Serververbindungskanal:
   a. Starten Sie den Warteschlangenmanager.
   b. Definieren Sie einen Beispielkanal namens NET.CHANNEL[3]:

      ```
      DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for WebSphere MQ classes for .NET')
      ```

2. Starten Sie einen Listener:

   ```
   runmqlsr -t tcp [-m qmnqme] [-p portnum]
   ```

[3] In diesem Beispiel werden keine Auswirkungen auf die Sicherheit berücksichtigt. In einem Produktionssystem können Sie SSL oder einen Sicherheitsexit verwenden. Weitere Informationen finden Sie im Abschnitt Sicherheit.

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.

Beispielanwendungen

Wenn Sie eigene .NET-Anwendungen ausführen möchten, folgen Sie den Anweisungen für die Prüfprogramme und ersetzen Sie dabei den Namen der Beispielanwendung durch den Namen Ihrer Anwendung.

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 von .NET unterstützten Sprache schreiben.

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 erklärt, wie WebSphere MQ Publish/Subscribe verwendet wird. 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 WebSphere 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 in 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
```

WebSphere MQ .NET-Probleme beheben

Wenn ein Programm nicht erfolgreich beendet wird, führen Sie eine der Beispielanwendungen aus und halten Sie sich an die Ratschläge in den Diagnosenachrichten.

Diese Beispielanwendungen werden unter „WebSphere MQ-Klassen für .NET verwenden” auf Seite 607 beschrieben.

Wenn die Fehler weiterhin auftreten und Sie den IBM Kundendienst kontaktieren müssen, werden Sie möglicherweise dazu aufgefordert, die Tracefunktion zu aktivieren.

Traceerstellung für die Beispielanwendung

Anweisungen zur Verwendung der Tracefunktion finden Sie im Abschnitt „Traceerstellung für WebSphere MQ .NET-Programme” auf Seite 630.

Fehlermeldungen

Es werden möglicherweise die folgende allgemeine Fehlernachricht angezeigt:

Eine nicht behandelte Ausnahmebedingung vom Typ 'System.IO.FileNotFoundException' ist in einem unbekannten Modul aufgetreten

Wenn dieser Fehler bei 'amqmdnet.dll' oder 'amqmdxcs.dll' auftritt, müssen Sie entweder sicherstellen, dass beide Dateien im globalen Assembly-Cache registriert sind, oder eine Konfigurationsdatei erstellen, die auf die 'amqmdnet.dll'- und 'amqmdxcs.dll'-Gruppen verweist. Sie können den Inhalt des Assembly-Cache mit Hilfe von 'mscorcfg.msc', das zum Lieferumfang des .NET-Frameworks gehört, überprüfen und ändern.

Wenn das .NET-Framework bei der Installation von WebSphere MQ nicht verfügbar war, wurden die Klassen möglicherweise nicht im Global Assembly Cache registriert. Sie können den Registrierungsprozess mit folgendem Befehl manuell erneut ausführen:

```
amqidnet -c MQ_INSTALLATION_PATH\bin\amqidotn.txt -l logfile.txt
```

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

Informationen zu dieser Installation werden in die angegebene Protokolldatei geschrieben (in diesem Beispiel `logfile.txt`).

.NET-Programme von WebSphere MQ schreiben und implementieren

Wenn Sie WebSphere MQ-Klassen für .NET nutzen möchten, um auf WebSphere MQ-Warteschlangen zuzugreifen, schreiben Sie Programme in einer von .NET unterstützten Sprache, die Aufrufe enthält, die Nachrichten in WebSphere MQ-Warteschlangen einreihen und daraus abrufen.

Die WebSphere MQ-Dokumentation enthält nur Informationen zu den Sprachen C#, C++ und Visual Basic.

Diese Themensammlung stellt Informationen bereit, die Sie beim Schreiben von Anwendungen für die Interaktion mit WebSphere MQ-Systemen unterstützen sollen. Weitere Informationen über diese Klassen finden Sie im Abschnitt Klassen und Schnittstellen von WebSphere MQ .NET.

Unterschiede bei der Verbindung

Die Art und Weise Ihrer WebSphere MQ .NET-Programmierung hängt von den Verbindungsmodi ab, die Sie verwenden möchten.

Verwaltete Clientverbindungen

Bei Verwendung von WebSphere MQ-Klassen für .NET als verwalteten Client gibt es eine Reihe von Unterschieden zum standardmäßigen WebSphere MQ-MQI-Client.

Sie folgenden Funktionen sind für einen verwalteten Client nicht verfügbar:

- Kanalkomprimierung
- SSL-Unterstützung
- 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 verwaltete WebSphere MQ-Problemdiagnose zu verwenden, wird keiner der Parameter -i, -p, -s, -b oder -c des Befehls **strmqtrc** unterstützt.

Eine verwaltete .NET-Anwendung, die XA-Transaktionen verwendet, 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 setzen, um den Verbindungstyp explizit auszuwählen (siehe folgende Tabelle).

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 Version 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 WebSphere MQ .NET eine Ausnahme aus.

Wenn Sie NMQ_MQ_LIB auf "managed" setzen, verwendet der Client verwaltete Diagnosetests für WebSphere MQ-Probleme, die .NET-Datenkonvertierung und weitere verwaltete und untergeordnete WebSphere MQ-Funktionen.

Bei allen anderen Werten für NMQ_MQ_LIB verwendet der .NET-Prozess Tests der nicht verwalteten WebSphere MQ-Problemdiagnose, die Datenkonvertierung und weitere nicht verwaltete und untergeordnete WebSphere MQ-Funktionen (vorausgesetzt, ein WebSphere MQ-MQI-Client oder -Server ist auf dem System installiert).

Der Verbindungstyp wird von WebSphere MQ .NET wie folgt ausgewählt:

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 nicht auf "managed" gesetzt ist, werden Diagnosetests für Probleme, die Datenkonvertierung und weitere untergeordnete Funktionen nicht vollständig verwaltet (vorausgesetzt, ein WebSphere 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 WebSphere MQ-Server installiert, wird eine Verbindung als Server hergestellt.
6. Ist ein WebSphere MQ-MQI-Client installiert, wird eine Verbindung als Nicht-XA-Client hergestellt.
7. Andernfalls wird die Verbindung als verwalteter Client hergestellt.

Konfigurationsdateien für WebSphere MQ-Klassen für .NET

Eine .NET-Clientanwendung kann eine WebSphere MQ-MQI-Clientkonfigurationsdatei und, falls Sie den Typ einer verwalteten Verbindung verwenden, eine .NET-Anwendungskonfigurationsdatei verwenden. Die Einstellungen in der Anwendungskonfigurationsdatei haben Priorität.

Clientkonfigurationsdatei

Eine WebSphere MQ-Klassen für .NET-Clientanwendung kann eine Clientkonfigurationsdatei auf die gleiche Weise wie jeder andere WebSphere MQ-MQI-Client verwenden. Diese Datei hat normalerweise die Bezeichnung 'mqclient.ini', aber Sie können einen anderen Dateinamen angeben. Weitere Informationen zur Clientkonfigurationsdatei finden Sie unter Client mit einer Konfigurationsdatei konfigurieren WebSphere MQ MQI-Clientkonfigurationsdatei.

Für WebSphere MQ-Klassen für .NET sind nur die folgenden Attribute in einer WebSphere MQ-MQI-Clientkonfigurationsdatei relevant. Wen Sie andere Attribute angeben, hat dies keine Auswirkungen.

Zeilengruppe	Attribut
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ServerConnectionParms
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 WebSphere MQ-Clientkonfigurationsdatei und die entsprechenden Umgebungsvariablen überschreiben.

Die Einstellungen der .NET-Anwendungskonfigurationsdatei werden nur bei 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 speziell auf WebSphere MQ zugeschnitten.

Beim Format der .NET-Anwendungskonfigurationsdatei handelt es sich um eine Anzahl von *Abschnitten*. Jeder Abschnitt enthält mindestens einen *Schlüssel* und jedem Schlüssel ist ein *Wert* zugeordnet. Im folgenden Beispiel werden die Abschnitte, Schlüssel und Werte gezeigt, die in einer .NET-Anwendungskonfigurationsdatei zur Steuerung der Eigenschaft für das TCP/IP-Keepalive verwendet werden:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Die Schlüsselwörter, die in den Abschnittsnamen und Schlüsseln der .NET-Anwendungskonfigurationsdatei verwendet werden, stimmen exakt mit den Schlüsselwörtern für die Zeilengruppen und Attribute überein, die in der Clientkonfigurationsdatei definiert sind.

Weitere Informationen finden Sie in Ihrer Microsoft-Dokumentation.

Beispielcodefragment

Das folgende C#-Codefragment zeigt eine Anwendung, die drei Aktionen ausführt:

1. Verbindung zu einem Warteschlangenmanager herstellen
2. Nachricht in SYSTEM.DEFAULT.LOCAL.QUEUE einreihen
3. Nachricht zurückerhalten

Es wird auch gezeigt, die der Verbindungstyp geändert wird.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
```

```
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define a WebSphere 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 a WebSphere 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 a WebSphere MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("A WebSphere 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
```

Operationen bei Warteschlangenmanagern

In diesem Abschnitt wird beschrieben, wie die Verbindung zu einem Warteschlangenmanager mit WebSphere MQ-Klassen für .NET hergestellt und wieder getrennt wird.

WebSphere MQ-Umgebung einrichten

Vor Verwendung der Clientverbindung für die Verbindung zu einem Warteschlangenmanager müssen Sie die WebSphere MQ-Umgebung einrichten.

Anmerkung: Dieser Schritt ist nicht erforderlich, wenn Sie WebSphere MQ-Klassen für .NET im Serververbindungsmodus verwenden.

Die .NET-Programmierschnittstelle ermöglicht die Verwendung des NMQ_MQ_LIB-Anpassungswerts und schließt eine Klasse namens "MQEnvironment" mit ein. Mit dieser Klasse können Sie Einzelheiten angeben, die während des Verbindungsversuchs verwendet werden sollen, wie beispielsweise die Einträge in der folgenden Liste:

- Kanalname
- Hostname
- Portnummer
- Kanalexits
- SSL-Parameter
- Benutzer-ID und Kennwort

Vollständige Informationen zur MQEnvironment-Klasse finden Sie unter `MQEnvironment` .NET-Klasse .

Verwenden Sie zur Angabe des Kanalnamens und des Hostnamens folgenden Code:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Die Clients versuchen standardmäßig, eine Verbindung zu einem WebSphere MQ-Listener an Port 1414 aufzubauen. Wenn Sie einen anderen Port angeben möchten, verwenden Sie folgenden Code:

```
MQEnvironment.Port = nnnn;
```

Verbindung mit einem Warteschlangenmanager herstellen

Sie können nun eine Verbindung zu einem Warteschlangenmanager herstellen, indem Sie eine neue Instanz der MQQueueManager-Klasse erstellen:

```
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 WebSphere MQ .NET können Sie auch eine Warteschlange mit dem MQQueue-Konstruktor 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. Anders als bei der prozeduralen Schnittstelle, wo mit MQPUT und MQGET Bytegruppen eingereiht und abgerufen werden, werden mit WebSphere MQ-Klassen für .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.

Wenn die neue Nachrichteninstanz erstellt wird, werden alle MQMD-Parameter automatisch auf ihre Standardwerte gesetzt, wie in Anfangswerte und Sprachendeklarationen 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 in .NET-Klasse MQMessage.

Anmerkung: Die WriteUTF()-Methode von MQMessage verschlüsselt automatisch die Länge der Zeichenfolge und die enthaltenen Unicode-Bytes. Wenn Ihre Nachricht von einem anderen .NET-Programm gelesen wird (unter Verwendung von ReadUTF()), ist dies die einfachste Möglichkeit, um Informationen zur Zeichenfolge 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. Zudem vereinfachen Nachrichteneigenschaften die Kommunikation zwischen WebSphere MQ- und JMS-Anwendungen. Weitere Informationen zu Nachrichteneigenschaften in WebSphere 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
- Lang
- Objekt
- Kurz
- Zeichenfolge

Um beispielsweise die WebSphere MQ-Eigenschaft 'myproperty' (eine Zeichenfolge) abzurufen, verwenden Sie den Aufruf `message.GetStringProperty('myproperty')`. Optional können Sie auch einen Eigenschaftsdeskriptor übergeben, der von WebSphere MQ abgeschlossen wird.

Fehler bearbeiten

Fehler, die aufgrund von WebSphere MQ-Klassen für .NET entstehen, werden mit `try`- und `catch`-Blöcken behandelt.

Die Methoden in der .NET-Schnittstelle geben weder einen Beendigungscode noch einen Ursachencode zurück. Stattdessen lösen sie eine Ausnahmebedingung aus, wenn Beendigungscode und Ursachencode nach einem WebSphere MQ-Aufruf nicht beide null sind. Dies vereinfacht die Programmlogik, sodass Sie nicht nach jedem WebSphere MQ-Aufruf die Rückgabecodes überprüfen 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 AccessQueue()-Method verwenden, um auf

ein Warteschlangenobjekt zuzugreifen, wird dieses Objekt *nicht* automatisch für die Inquire- oder Set-Operationen 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 ist eine Multithread-Umgebung. In WebSphere MQ-Klassen für .NET kann ein Warteschlangenmanagerobjekt von mehreren Threads gemeinsam verwendet werden; gleichzeitig stellen WebSphere MQ-Klassen für .NET sicher, dass der gesamte Zugriff auf den Zielwarteschlangenmanager synchronisiert wird.

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).

Mit der Implementierung von WebSphere MQ .NET wird sichergestellt, dass der Zugriff einer bestimmten Verbindung (MQQueueManager-Objektinstanz) auf den WebSphere MQ .NET-Zielwarteschlangenmanager synchronisiert ist. Standardverhalten: Ein Thread, der einen Aufruf an einen Warteschlangenmanager absetzen will, wird blockiert, bis alle anderen aktiven Aufrufe für diese Verbindung abgearbeitet 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 Clientkanal mit .NET verwenden

Sie können eine Definitionstabelle für einen Clientkanal mit den .NET-Klassen für WebSphere 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 mehrere Arten angegeben ist, dann haben die Umgebungsvariablen Vorrang vor der Clientkonfigurationsdatei und die .NET-Anwendungskonfigurationsdatei hat Vorrang 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.

Vorgehensweise einer .NET-Anwendung beim Ermitteln der zu verwendenden Kanaldefinition

In der WebSphere 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 WebSphere MQ verwendenNETZ

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

Für eine verwaltete Verbindung definieren Sie zur Implementierung eines Exits eine neue .NET-Klasse, die die entsprechende Schnittstelle implementiert. Im WebSphere 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 Rückgabe muss der Wert von *dataOffset* und *dataLength* auf das Offset und die Länge innerhalb der zurückgegebenen Bytefeldgruppe verweisen, die die .NET-Klassen dann verwenden sollen. 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'

V 7.5.0.6 Ab Version 7.5.0, Fix Pack 6 werden die mit der verwalteten .NET-Clientanwendung angegebene Benutzer-ID und das Kennwort in der, IBM WebSphere MQ-Klasse .NET 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 432).

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 Clientkanal mit .NET verwenden" auf Seite 621.

Die Spezifikation weist immer die Form einer Zeichenfolge mit folgendem Format auf:

```
Assembly_name(Class_name)
```

Class_name ist der vollständig qualifizierte Name (einschließlich Namensbereichsangabe) einer .NET-Klasse, die die IBM.WMQ.MQSecurityExit-, IBM.WMQ.MQSendExit- oder IBM.WMQ.MQReceiveExit-Schnittstelle implementiert (soweit 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. Wenn erforderlich, wird vom .NET-Client-Code durch Analysieren der Zeichenfolgespezifikation eine Instanz der angegebenen Klasse geladen und erstellt.

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 Clientkanal mit .NET verwenden" auf Seite 621.

Bei der Spezifikation handelt es sich in jedem Fall um eine Zeichenfolge, die auf 32 Zeichen beschränkt ist.

Automatische Clientverbindungswiederholung 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 WebSphere MQ .NET-Client handelt, kann für ihn die automatische Clientverbindungswiederholung gewählt werden, indem für CONNECT_OPTIONS_PROPERTY ein entsprechender Wert angegeben wird, wenn Sie einen Warteschlangenmanager mithilfe der Klasse `MQQueueManager` erstellen. 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).

Secure Sockets Layer (SSL)-Unterstützung

Der folgende Abschnitt gilt nicht für verwaltete Clients.

WebSphere MQ-Klassen für .NET-Clientanwendungen unterstützen die SSL-Verschlüsselung (SSL = Secure Sockets Layer). SSL stellt eine Kommunikationsverschlüsselung, Authentifizierung und Nachrichtenintegrität bereit. In der Regel wird damit die Kommunikation zwischen zwei Peers im Internet oder innerhalb eines Intranets geschützt.

SSL aktivieren

SSL wird nur für Clientverbindungen unterstützt. Zum Aktivieren von SSL müssen Sie die CipherSpec angeben, die bei der Kommunikation mit dem Warteschlangenmanager verwendet wird. Diese muss außerdem mit der CipherSpec übereinstimmen, die im Zielkanal festgelegt ist.

Zum Aktivieren von SSL geben Sie die CipherSpec mithilfe der statischen Mitgliedsvariablen 'SSLCipherSpec' von MQEnvironment an. Das folgende Beispiel bezieht sich auf einem SVRCONN-Kanal mit der Bezeichnung SECURE.SVRCONN.CHANNEL, der konfiguriert wurde, um SSL mit dem CipherSpec NULL_MD5 anzufordern:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "NULL_MD5";
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.

Um eine erfolgreiche Verbindung über SSL herzustellen, muss der Client-Keystore mit dem Stammzertifikat der Zertifizierungsstelle konfiguriert werden, aus dem das Zertifikat authentifiziert werden kann, das vom Warteschlangenmanager übergeben wird. 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 SSL-Zertifikat, das einen *Definierten Namen* (DN) enthält.

Eine WebSphere MQ .NET-Clientanwendung kann diesen definierten Namen verwenden, um sicherzustellen, dass sie Daten mit dem richtigen Warteschlangenmanager austauscht. Ein definiertes Namensmuster wird mit der Variablen 'sslPeerName' von MQEnvironment angegeben. Sie können beispielsweise Folgendes festlegen:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Die Verbindung kann nur erfolgreich hergestellt werden, wenn der Warteschlangenmanager ein Zertifikat mit einem allgemeinen Namen vorlegt, der mit 'QMGR.' beginnt. und mindestens zwei Organisationseinheitennamen, von denen der erste IBM und der zweite WEBSPHERE sein muss.

Die Eigenschaft 'SSLPeerName' kann auch mit MQC.SSL_PEER_NAME_PROPERTY in der Hashtabelle mit Verbindungseigenschaften festgelegt werden. Weitere Informationen zu definierten Namen und Regeln zum Festlegen von Peernamen finden Sie im Abschnitt Sicherheit.

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.

Fehlerbearbeitung bei Verwendung von SSL

Die folgenden Ursachencodes können von WebSphere MQ-Klassen für .NET beim Aufbau der Verbindung zu einem Warteschlangenmanager über SSL ausgegeben werden:

MQRC_SSL_NOT_ALLOWED
Die Eigenschaft 'SSLCipherSpec' wurde festgelegt, aber die Verbindung über Bindungen wurde verwendet. Nur Clientverbindungen unterstützen SSL.

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.

.NET-Monitor verwenden

Wichtige Informationen finden Sie unter Features, die nur mit der primären Installation unter Windows verwendet werden können .

.NET Monitor ist eine Anwendung, die mit einem WebSphere MQ-Auslösemonitor vergleichbar ist. Sie können .NET-Komponenten erstellen, die bei jedem Empfang einer Nachricht in einer überwachten Warteschlange instanziiert werden und anschließend diese Nachricht verarbeiten. .NET Monitor wird mit dem Befehl `runmqdnm` gestartet und mit dem Befehl `endmqdnm` gestoppt. Einzelheiten zu diesen Befehlen finden Sie unter runmqdnm und endmqdnm.

Wenn Sie .NET Monitor verwenden möchten, schreiben Sie eine Komponente, die die IMQObjectTrigger-Schnittstelle implementiert, die in amqmdnm.dll definiert ist.

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`. Es kann auch eine Benutzerparameterzeichenfolge empfangen, wenn eine angegeben wurde, mit der Befehlszeilenoption *-u* , wenn `runmqdnm` gestartet 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 dann 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 vollständige Kontextinformationen für diese Nachricht und kann beispielsweise eine andere Warteschlange auf demselben Warteschlangenmanager öffnen, ohne eine eigene Verbindung mit WebSphere MQ herstellen zu müssen.

Beispielcodefragmente

Dieser Abschnitt enthält zwei Beispiele für Komponenten, die eine Nachricht von .NET Monitor erhalten und diese drucken; die eine mittels transaktionsorientierter Verarbeitung und die andere mittels nicht

transaktionsorientierter Verarbeitung. In einem dritten Beispiel werden Dienstprogrammroutinen gezeigt, die für die ersten beiden Beispiele angewendet werden können. Alle Beispiele sind in C# geschrieben.

Beispiel 1: Transaktionsorientierte Verarbeitung

```
/********************************************************************/
/* Licensed materials, property of IBM                              */
/* 63H9336                                                          */
/* (C) Copyright IBM Corp. 2005, 2024.                              */
/********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

Beispiel 2: Nicht transaktionsorientierte Verarbeitung

```
/********************************************************************/
/* Licensed materials, property of IBM                              */
/* 63H9336                                                          */
/* (C) Copyright IBM Corp. 2005, 2024.                              */
/********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c NonTran
namespace dnmsamp
```

```
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

Beispiel 3: Allgemeine Routinen

```
/**********************************************************************/
/* Licensed materials, property of IBM                                */
/* 63H9336                                                            */
/* (C) Copyright IBM Corp. 2005, 2024.                                */
/**********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ------------------------------------------------------------------ */
    /* Default prefix string of the namespace.                             */
    /* ------------------------------------------------------------------ */
    private string prefixText = "dnmsamp";

    /* ------------------------------------------------------------------ */
    /* Constructor that takes the replacement prefix string to use.       */
    /* ------------------------------------------------------------------ */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ------------------------------------------------------------------ */
    /* Display an arbitrary string to the console.                        */
    /* ------------------------------------------------------------------ */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
```

```
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }

    }
}
```

WebSphere MQ .NET-Programme kompilieren

Beispielbefehle für die Kompilierung von in verschiedenen Sprachen geschriebenen .NET-Anwendungen.

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

Geben Sie folgenden Befehl ein, um eine Anwendung in C# unter Verwendung von WebSphere MQ-Klassen für .NET zu erstellen:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib:MQ_INSTALLATION_PATH\bin /out:MyProg.exe MyProg.cs
```

Geben Sie folgenden Befehl ein, um eine Anwendung in Visual Basic unter Verwendung von WebSphere MQ-Klassen für .NET zu erstellen:

```
vbc /r:System.dll /r:MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Geben Sie folgenden Befehl ein, um eine verwaltete Anwendung in C++ unter Verwendung von WebSphere MQ-Klassen für .NET zu erstellen:

```
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.

Traceerstellung für WebSphere MQ .NET-Programme

In WebSphere MQ .NET starten und steuern Sie die Tracefunktion wie in WebSphere MQ-Programmen mithilfe der MQI.

Die Parameter -i und -p des Befehls 'strmqtrc', mit denen Sie Prozess-IDs, Thread-IDs und benannte Prozesse angeben können, haben allerdings keine Auswirkung.

Normalerweise führen Sie die Tracefunktion nur auf Anforderung des IBM Kundendienstes aus.

Weitere Informationen zu Tracebefehlen finden Sie in Trace unter Windows verwenden.

Angepasster IBM WebSphere MQ-Kanal für Microsoft Windows Communication Foundation (WCF)

Der angepasste WCF-Kanal (WCF = Microsoft Windows Communication Foundation) für IBM WebSphere MQ sendet und empfängt Nachrichten zwischen WCF-Clients und -Services.

Zugehörige Konzepte

„Einführung in die Verwendung des angepassten WebSphere MQ-Kanals für WCF mit .NET 3" auf Seite 631
Übersicht über die Informationen, die für Programmierer verfügbar sind, die den angepassten WebSphere MQ-Kanal für Windows Communication Foundation (WCF) mit .NET 3 verwenden.

„Angepasste WebSphere MQ-Kanäle für WCF verwenden" auf Seite 635
Übersicht zu den Informationen, die für Programmierer verfügbar sind, die die angepassten Kanäle von WebSphere MQ V7 für Windows Communication Foundation (WCF) verwenden.

„WCF-Beispiele verwenden" auf Seite 654
Die Beispiele für Windows Communication Foundation (WCF) stellen einige einfache Beispiele zur Verwendung des angepassten WebSphere MQ-Kanals bereit.

„Problembestimmung im angepassten WCF-Kanal für WebSphere MQ" auf Seite 660
Mit dem WebSphere MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des WebSphere 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 WebSphere MQ-Kanals für WCF mit .NET 3

Übersicht über die Informationen, die für Programmierer verfügbar sind, die den angepassten WebSphere MQ-Kanal für Windows Communication Foundation (WCF) mit .NET 3 verwenden.

Informationen zum angepassten WebSphere MQ-Kanal für WCF

Beim angepassten Kanal für WebSphere 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 die von den Transportmethoden und Protokollen, die zur Verbindungsherstellung verwendet werden, unabhängige Entwicklung. Es werden alternative Transportmethoden und Konfigurationen zur Verwendung aktiviert, die sich nach der Umgebung richten, in der der Service oder die Anwendung implementiert wird.

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 WebSphere 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 An-

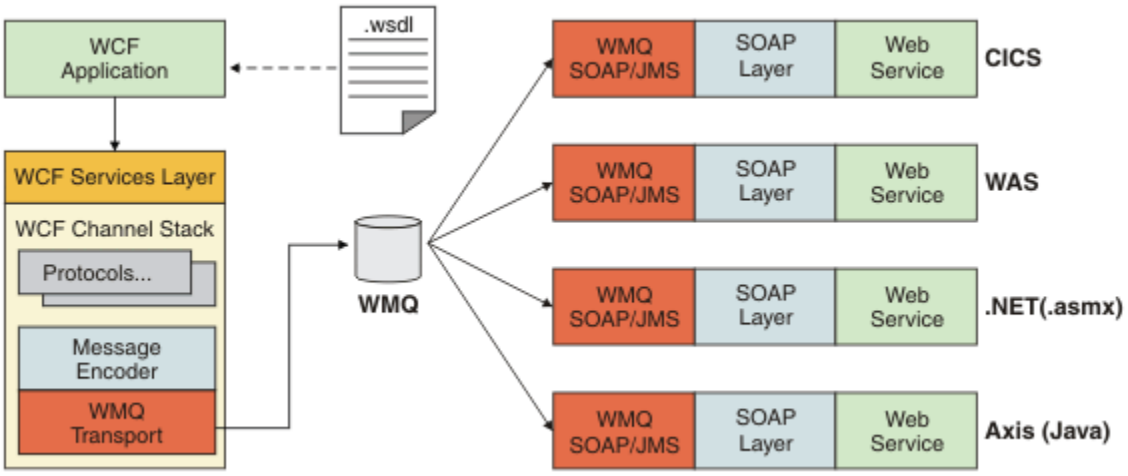
wendungen, die für die Verwendung von WCF entwickelt wurden, den angepassten Kanal für WebSphere MQ verwenden, um Daten auf die gleiche Weise zu senden und zu empfangen, wie dies bei der Verwendung von integrierten Transporten geschieht, die von Microsoft bereitgestellt werden. Dies ermöglicht die einfache Integration in die asynchronen, skalierbaren und zuverlässigen Funktionen zur Nachrichtenübermittlung von WebSphere MQ. Eine vollständige Liste der unterstützten Funktionen finden Sie unter „Funktionen und Leistungsmerkmale des angepassten WCF-Kanals” auf Seite 635.

Wann und warum wird der angepasste WebSphere MQ-Kanal für WCF verwendet?

Über den angepassten WebSphere MQ-Kanal können Sie Nachrichten zwischen WCF-Clients und -Services auf dieselbe Weise senden und empfangen wie über die von Microsoft bereitgestellten integrierten Transportprotokolle. Anwendungen erhalten somit Zugriff auf die Funktionen von WebSphere MQ innerhalb des einheitlichen WCF-Programmiermodells.

Ein typisches Szenario für das Verwendungsmuster des angepassten WebSphere MQ-Kanals für WCF ist die Verwendung als Schnittstelle für Web-Services, die über WebSphere MQ (SOAP/JMS) gehostet werden.

Nachrichten werden unter Verwendung des Nachrichtenformats SOAP over JMS von WebSphere MQübertragen, sodass WCF-Clients und -Services auch andere WebSphere MQ -Anwendungen oder -Hosting-Umgebungen aufrufen oder von diesen aufgerufen werden können, die mit diesem Format kompatibel sind, einschließlich Web-Services und Clients, die in WebSphere Application Server, CICS, Axis v1 (Java) ausgeführt werden. und .asmx (.NET), wie im folgenden Diagramm dargestellt:



Detaillierte Informationen zu SOAP over JMS finden Sie in „WebSphere MQ-Transport für SOAP” auf Seite 1005.

Im Folgenden sehen Sie ein Beispiel für ein typisches Szenario aus dem Diagramm :

1. Ein Web-Service wird in WebSphere Application Server gehostet und ist mithilfe der Unterstützung von SOAP over JMS im WebSphere Application Server über WebSphere MQ zugänglich.
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 Verwendung alternativer Encoder kann außerdem eine eingeschränkte Integration in native WebSphere MQ-Anwendungen bereitgestellt werden, die SOAP over JMS nicht unterstützen. Dies ist jedoch 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.

Softwarevoraussetzungen und Installationsanweisungen für angepassten WebSphere MQ-Kanal für WCF

In diesem Abschnitt werden die Softwarevoraussetzungen und Installationsinformationen für den angepassten WebSphere MQ-Kanal für WCF zusammengefasst.

Der angepasste WebSphere MQ-Kanal für WCF kann nur eine Verbindung zu Warteschlangenmanagern von WebSphere MQ V7 oder höher herstellen.

Softwarevoraussetzungen für den angepassten WCF-Kanal für WebSphere MQ

In diesen Informationen werden die Softwarevoraussetzungen für den angepassten WCF-Kanal für WebSphere MQaufgelistet.

Laufzeitumgebung

- Auf der Hostmaschine muss Microsoft .NET Framework v3.0 oder höher installiert sein.
- *Java and .NET Messaging and Web Services* wird standardmäßig als Teil des Installationsprogramms von WebSphere MQ 7.0.1 installiert. Die .NET-Assemblys, die für den angepassten Kanal erforderlich sind, werden im Global Assembly Cache installiert.

Anmerkung: Wenn Microsoft .NET Framework v2.0 oder höher vor der Installation von WebSphere MQ V7.0.1 nicht installiert wurde, wird die WebSphere MQ-Produktinstallation ohne Fehler fortgesetzt, der angepasste WebSphere MQ-Kanal ist jedoch nicht verfügbar. Wenn .NET Framework nach der Installation von WebSphere MQ 7.0.1installiert wird, müssen Sie den angepassten WebSphere MQ -Kanal aktivieren, indem Sie das Script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd ausführen. Dabei steht *WMQInstallDir* für das Verzeichnis, in dem WebSphere MQ 7.0.1 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 auf v3.0 oder höher von einer früheren Version durchgeführt wird, beispielsweise von .NET v2.0.

Entwicklungsumgebung

- Microsoft Visual Studio 2008 oder Windows Software Development Kit für .NET 3.0 oder höher.
- Auf der Hostmaschine muss Microsoft .NET Framework V3.5 oder höher installiert sein, um die Musterlösungsdateien erstellen zu können.

Anmerkung: Wenn Microsoft .NET Framework v2.0 oder höher vor der Installation von WebSphere MQ V7.0.1 nicht installiert wurde, wird die WebSphere MQ-Produktinstallation ohne Fehler fortgesetzt, der angepasste WebSphere MQ-Kanal ist jedoch nicht verfügbar. Wenn .NET Framework nach der Installation von WebSphere MQ 7.0.1installiert wird, müssen Sie den angepassten WebSphere MQ -Kanal aktivieren, indem Sie das Script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd ausführen. Dabei steht *WMQInstallDir* für das Verzeichnis, in dem WebSphere MQ 7.0.1 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 auf v3.0 oder höher von einer früheren Version durchgeführt wird, beispielsweise von .NET v2.0.

Angepasster WebSphere MQ-Kanal für WCF: Was wird installiert?

Beim angepassten Kanal für WebSphere MQ handelt es sich um einen Transportkanal, der das einheitliche Programmiermodell von Microsoft Windows Communication Foundation (WCF) verwendet. Der angepasste Kanal wird standardmäßig als Teil der Installation von WebSphere MQ Version 7.0.1 installiert.

Angepasster WebSphere MQ-Kanal für WCF

Der angepasste WebSphere MQ für WCF wird standardmäßig im Rahmen der Installation von WebSphere MQ 7.0.1 installiert. Der angepasste Kanal und seine Abhängigkeiten sind in der Komponente `Java and .NET Messaging and Web Services` enthalten, die standardmäßig installiert wird. Wenn Sie ein Upgrade von einer früheren Version auf WebSphere MQ 7.0.1 durchführen, wird der angepasste WebSphere MQ -Kanal für WCF standardmäßig installiert, wenn die Komponente `Java and .NET Messaging and Web Services` zuvor in einer früheren Installation installiert wurde.

Die Komponente `Java and .NET Messaging and Web Services` enthält die Datei `IBM.XMS.WCF.dll` und die Datei `IBM.XMS.WCF.dll` ist die Hauptassembly des angepassten Kanals, die die WCF-Schnittstellenklassen enthält. Diese Datei ist im Global Assembly Cache (GAC) installiert und außerdem im folgenden Verzeichnis verfügbar: *MQ_INSTALLATION_PATH*`\bin`. Hierbei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem WebSphere MQ 7.0.1 installiert ist.

Die wichtigsten Klassen, die zur Verwendung des angepassten Kanals erforderlich sind, befinden sich im *Namensbereich: IBM.XMS.WCF* und:

Name der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement
Importkomponente der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter
Konfiguration der Transportbindung	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig

Beispiele für angepassten WebSphere MQ-Kanal

In den Beispielen werden einfache Verwendungsmöglichkeiten des angepassten WebSphere MQ-Kanals für WCF gezeigt. Die Beispiele und die zugehörigen Dateien befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\wcf\samples\`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von WebSphere MQ. Weitere Informationen zu den Beispielen für den angepassten WebSphere MQ-Kanal finden Sie in „WCF-Beispiele verwenden” auf Seite 654.

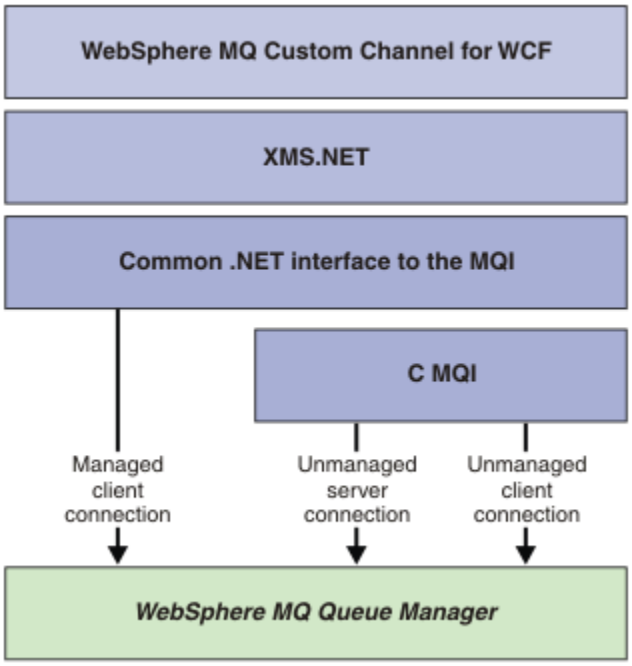
svcutil.exe.config

Die Datei `svcutil.exe.config` ist ein Beispiel für die Konfigurationseinstellungen, die zur Aktivierung des Microsoft-WCF-Generierungstools `svcutil` für den Client-Proxy zur Erkennung des angepassten Kanals erforderlich sind. Die Datei `svcutil.exe.config` befindet sich im Verzeichnis *MQ_INSTALLATION_PATH*`\tools\wcf\docs\examples\`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von WebSphere MQ. Weitere Informationen zur Verwendung der Datei `svcutil.exe.config` finden Sie in „WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit Metadaten aus einem aktiven Service generieren” auf Seite 651.

WCF-Architektur

Der angepasste WebSphere MQ -Kanal für WCF ist über die `IBM Message Service Client for .NET (XMS .NET)` -API integriert.

Die WCF-Architektur wird im folgenden Diagramm gezeigt:



Alle erforderlichen Komponenten werden standardmäßig mit der Installation von WebSphere MQ V7.0.1 installiert.

Es stehen drei Verbindungen zur Verfügung: Verwaltete Clientverbindungen, nicht verwaltete Serververbindungen und nicht verwaltete Clientverbindungen. Weitere Informationen zu diesen Verbindungen finden Sie unter „Optionen für WCF-Verbindungen” auf Seite 640.

Angepasste WebSphere MQ-Kanäle für WCF verwenden

Übersicht zu den Informationen, die für Programmierer verfügbar sind, die die angepassten Kanäle von WebSphere MQ V7 für Windows Communication Foundation (WCF) verwenden.

Die Microsoft Windows Communication Foundation unterstützt die Web-Services und die Messaging-Unterstützung in Microsoft .NET Framework 3. WebSphere MQ V7 kann jetzt als angepasster Kanal 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 WebSphere MQ V7 formatiert. Anwendungen können anschließend mit Services kommunizieren, die von WCF oder von der Serviceinfrastruktur von WebSphere SOAP over JMS gehostet werden. Detaillierte Informationen zu SOAP over JMS finden Sie in „WebSphere MQ-Transport für SOAP” auf Seite 1005.

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 zu den Formen des angepassten Kanals, mit denen WebSphere MQ wie in den angepassten Kanälen von Microsoft Windows Communication Foundation (WCF) verwendet werden kann.

Der angepasste WebSphere 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 WebSphere 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 WebSphere MQ-Kanal für Anforderungen und Antworten 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

connectionFactory
Der Parameter 'connectionFactory' ist erforderlich. Informationen zur Syntax dieses Parameters finden Sie unter URI-Syntax und -Parameter für die Web-Service-Implementierung .

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 1066).

Zuverlässige Nachrichtenübermittlung mit dem angepassten WCF-Kanal

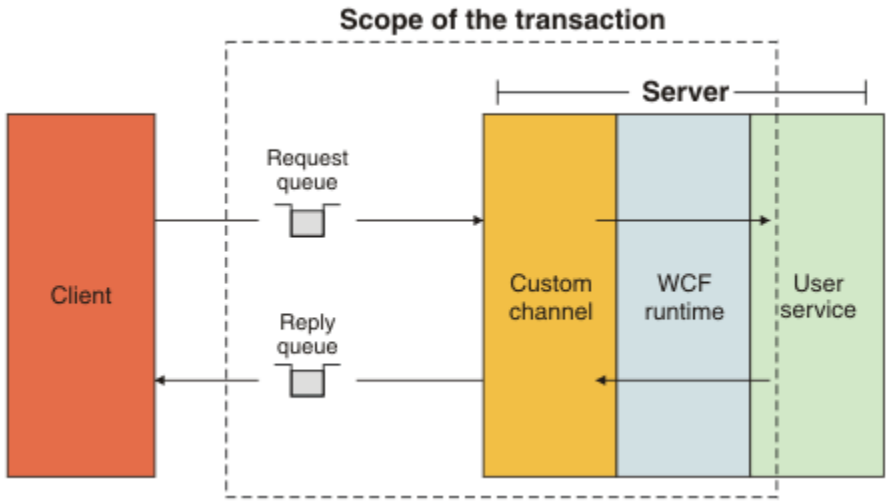
Die zuverlässige Nachrichtenübermittlung garantiert, dass eine Serviceanforderung oder -antwort ausgeführt wird und nicht verloren geht.

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 unter URI-Syntax und -Parameter für die Web-Service-Implementierung.

Die dezentrale Transaktionsverarbeitung wird nicht unterstützt und der Bereich der Transaktion reicht nicht über die Verarbeitung von Anforderungs- und Antwortnachrichten hinaus, die von WebSphere MQ ausgeführt wird. 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 638.
- 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 646.

- Bei der Funktion für die zuverlässige Nachrichtenübermittlung werden die WCF-Erweiterbarkeitspunkte 'IOperationInvoker' und 'IErrorHandler' verwendet. Wenn diese Erweiterbarkeitspunkte extern von einer Anwendung verwendet werden, muss die Anwendung sicherstellen, dass alle zuvor registrierten Erweiterbarkeitspunkte aufgerufen werden. Wenn dies für 'IErrorHandler' nicht geschieht, werden Fehler möglicherweise nicht gemeldet. Wenn dies für 'IOperationInvoker' nicht geschieht, antwortet der WCF-Service möglicherweise nicht mehr.

Sicherheit für den angepassten WCF-Kanal

Der angepasste WebSphere MQ-Kanal für WCF unterstützt die Verwendung von SSL nur für nicht verwaltete Clientverbindungen zum Warteschlangenmanager.

SSL kann auf zwei Arten angegeben werden:

- Angabe von SSL direkt im URI für SOAP over JMS. Eine vollständige Beschreibung der SSL-Optionen finden Sie in SSL und der WebSphere MQ-Transport für SOAP.
- Angabe von SSL mithilfe eines Eintrags in der Definitionstabelle für den Clientkanal (CCDT). Weitere Informationen zur Definitionstabelle für den Clientkanal finden Sie in Definitionstabelle für den Clientkanal.

Definitionstabellen für den Clientkanal (CCDT) in WCF

Der angepasste WebSphere 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 im 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 Definitionstabellen für Clientkanäle finden Sie unter Definitionstabelle für Clientkanäle .

Zugehörige Konzepte

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 636
- 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:

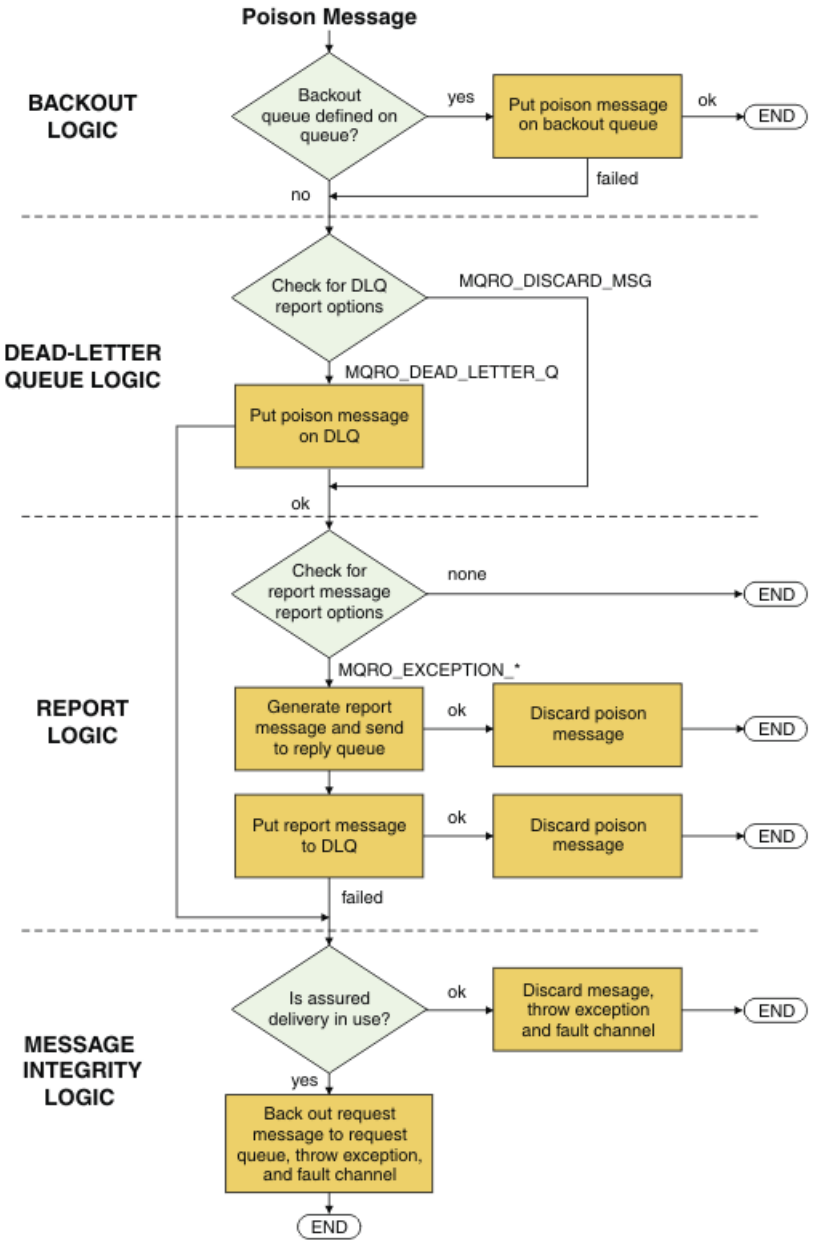
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

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:



Optionen für WCF-Verbindungen

Ein angepasster WebSphere 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 611

Weitere Informationen zur WCF-Architektur finden Sie unter „WCF-Architektur” auf Seite 634

Nicht verwaltete Clientverbindung

Bei diesem Verbindungsmodus wird ein WebSphere MQ-Client mit einem WebSphere MQ-Server verbunden, der auf einem lokalen oder fernen System ausgeführt wird.

Um den angepassten WebSphere MQ-Kanal für WCF als WebSphere MQ-Client zu verwenden, können Sie ihn mit dem WebSphere MQ-MQI-Client entweder auf dem WebSphere MQ-Server oder auf einem separaten System installieren.

Nicht verwaltete Serververbindung

Im Serverbindungsmodus kommuniziert der angepasste WebSphere MQ-Kanal für WCF nicht über ein Netz, sondern verwendet die Warteschlangenmanager-API. Durch die Verwendung von Verbindungen über Bindungen wird die Leistung für WebSphere MQ-Anwendungen im Vergleich zur Verwendung von Netzverbindungen verbessert.

Zur Verwendung von Verbindungen über Bindungen müssen Sie den angepassten WebSphere MQ-Kanal für WCF auf dem WebSphere MQ-Server installieren.

Verwaltete Clientverbindung

Bei diesem Verbindungsmodus wird ein WebSphere MQ-Client mit einem WebSphere MQ-Server verbunden, der auf einem lokalen oder fernen System ausgeführt wird.

Die Klassen für .NET 3 des angepassten WebSphere MQ-Kanals, die eine Verbindung in diesem Modus herstellen, verbleiben im von .NET verwalteten Code und rufen keine nativen Services auf. Weitere Informationen zum verwalteten 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 611.

Angepassten WebSphere MQ-Kanal für WCF erstellen und konfigurieren

Die angepassten Kanäle von WebSphere MQ V7 für WCF funktionieren auf die gleiche Weise wie WCF-Kanäle für den Transport, die von Microsoft angeboten werden. Der angepasste WebSphere MQ-Kanal für WCF kann auf zwei Arten erstellt werden.

Informationen zu diesem Vorgang

Der angepasste WebSphere MQ-Kanal wird als WCF-Transportkanal in WCF integriert und muss deshalb mit einem Nachrichtenencoder und optionalen Protokollkanälen paarweise verbunden werden, damit ein vollständiger 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. Für den angepassten Kanal wird ein URI im SOAP over JMS-URI-Format verwendet.

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 WebSphere 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 WebSphere MQ-Kanals für WCF, bei dem es sich um einen WCF-Kanal auf Transportebene handelt, müssen eine Bindung und eine Endpunktdefinition festgelegt werden. 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 644).
- 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 mit der rechten Maustaste auf **Erweitert** > **Erweiterung** > **Bindungselement** , um das Menü zu öffnen, und wählen Sie **Neu** aus.
2. Füllen Sie die Felder wie in dieser Tabelle gezeigt aus:

Tabelle 73. 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 74. Felder für neue Bindungskonfigurationen

Feld	**Wert**
Name	`CustomBinding_WMQ`
Bindungselement 1	`textMessageEncoding (MessageVersion: Soap11)`

Tabelle 74. Felder für neue Bindungskonfigurationen (Forts.)	
Feld	**Wert**
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 642 erstellt haben.
2. Nehmen Sie alle erforderlichen Änderungen an den Standardwerten der Eigenschaften wie in „Optionen zur WCF-Bindungskonfiguration" auf Seite 646 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 642 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 75. 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 „Angepasster WebSphere MQ-Kanal für WCF - Adressformat für WCF-Endpunkt" auf Seite 645
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 76. Felder für neue Serviceendpunkte	
Feld	**Wert**
Name	`Endpoint_WMQ`

Tabelle 76. Felder für neue Serviceendpunkte (Forts.)	
Feld	**Wert**
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 „Angepasster WebSphere MQ-Kanal für WCF - Adressformat für WCF-Endpunkt” auf Seite 645
Bindung	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Vertrag	*Name der Serviceimplementierungsklasse*

Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen

Beim angepassten WebSphere 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 WebSphere MQ-Kanals für WCF, bei dem es sich um einen WCF-Kanal auf Transportebene handelt, müssen eine Bindung und eine Endpunktdefinition festgelegt werden. Die Bindung enthält die Konfigurationsinformationen für den Kanal und die Endpunktdefinition enthält die Verbindungsdetails. Weitere Informationen finden Sie in „WCF-Beispiele verwenden” auf Seite 654.

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 642).
- Programmgesteuert direkt aus dem Anwendungscode (siehe folgendes Beispiel).

Schritt 1: Erstellen Sie eine Instanz des Transportbindungselements des Kanals

Fügen Sie Ihrer Anwendung den folgenden Code hinzu:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new SoapJmsIbmTransportBindingEle
ment();
```

Schritt 2: Legen Sie die Bindungseigenschaften fest

Definieren Sie alle erforderlichen Bindungseigenschaften. Fügen Sie beispielsweise folgenden Code zu Ihrer Anwendung hinzu, um den Clientverbindungsmodus (`ClientConnectionMode`) festzulegen.

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

Schritt 3: Erstellen Sie eine benutzerdefinierte Bindung, die den Transportkanal mit einem Nachrichtenencoder verknüpft

Erstellen Sie eine benutzerdefinierte Bindung, indem Sie den folgenden Code zu Ihrer Anwendung hinzufügen:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(), transportBinding
Element);
```

Schritt 4: Erstellen des URI für SOAP/JMS

Der URI für SOAP/JMS, der die WebSphere MQ-Verbindungsdetails beschreibt, die für den Zugriff auf den Service erforderlich sind, muss als Endpunktadresse angegeben werden. Hierbei spielt es eine Rolle, ob der Kanal für eine Serviceanwendung oder eine Clientanwendung benutzt wird.

Bei Clientanwendungen muss der URI für SOAP/JMS als Endpunktadresse (`EndpointAddress`) wie folgt erstellt werden:

```
EndpointAddress address = new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFac
tory=connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

Bei Serviceanwendungen muss der URI für SOAP/JMS als URI wie folgt erstellt werden:

```
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=connectQueueMana
ger(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

Weitere Informationen zur Endpunktadresse finden Sie in „Angepasster WebSphere MQ-Kanal für WCF - Adressformat für WCF-Endpunkt” auf Seite 645.

Angepasster WebSphere MQ-Kanal für WCF - Adressformat für WCF-Endpunkt

Ein URI (Universal Resource Identifier) stellt Einzelheiten zur Position und zur Verbindung bereit, die zur Angabe eines Web-Service benötigt werden. Dieses URI-Format ermöglicht einen umfassenden Steuerungsgrad für SOAP/ WebSphere MQ-spezifische Parameter und Optionen beim Zugriff auf Zielservices.

Ein Web-Service wird mit einem Universal Resource Identifier (URI) angegeben. In diesem Abschnitt wird das URI-Format erläutert, das in WebSphere MQ Transport for SOAP unterstützt wird. Dieses URI-Format erlaubt beim Zugriff auf die Zielservices über SOAP- und WebSphere MQ-spezifische Parameter und Optionen die umfassende Steuerung. Dieses Format ist mit WebSphere Application Server (WAS) und mit CICS kompatibel, was die Integration von WebSphere MQ mit beiden Produkten vereinfacht.

Die URI-Syntax lautet folgendermaßen:

```
jms:/queue?name=value&name=value...
```

Dabei ist `name` ein Parametername und *wert* ein geeigneter Wert und das Element `name`=*wert* kann beliebig oft wiederholt werden, wobei dem zweiten und nachfolgenden Vorkommen ein Et-Zeichen (&) vorangestellt wird.

Weitere Informationen zum Festlegen der URI-Eigenschaften finden Sie in URI-Syntax und -Parameter für die Web-Service-Implementierung.

Bei Parameternamen muss wie bei den Namen von WebSphere MQ-Objekten die Groß-/Kleinschreibung beachtet werden. 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 eingeschlossen werden, werden diese Parameter 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
```

Im Folgenden ist ein Beispiel eines einfachen URI für einen .NET-Service aufgeführt:

```
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 Achsenbeispiel 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 Option `sslPeerName` von `connectionFactory` 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
```

Optionen zur WCF-Bindungskonfiguration

In diesem Abschnitt wird beschrieben, wie Konfigurationsoptionen auf die Bindungsinformationen der angepassten Kanäle angewendet werden können. Des Weiteren werden die Optionen aufgelistet, die zur Verfügung stehen.

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 auch in der Anwendungskonfigurationsdatei angegeben werden, z. B. `app.config`. Die Konfigurationsdatei wird z. B. mit **`svcutil`** generiert.

```
<customBinding>

...

     <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>

...

</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 SoapJmsIbmTransportBindingEle
ment();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

WCF-Bindungseigenschaften

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
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.

Eigenschaftsname	Client- oder Service-anwen-dung	Administ-rativer Wert	Programmgesteuerter Wert	Beschreibung
MaxConcurrentCalls	Client	Der Bereich liegt bei 0 - 2 147 483 647 16 ist der Standard-wert	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.

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 636). 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 zu den Services für Microsoft Windows Communication Foundation (WCF), in der das Erstellen und Konfigurieren von WCF-Services erläutert wird.

Der angepasste IBM WebSphere MQ-Kanal für WCF und die WCF-Services, die diesen verwenden, können von den folgenden Methoden bereitgestellt werden:

- Self-Hosting
- Windows-Dienst

Der angepasste IBM WebSphere MQ-Kanal für WCF kann nicht im Windows-Prozessaktivierungsdienst gehostet 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 642 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 in *`WMQInstallDir`*`\bin` , wobei *WMQInstallDir* das Verzeichnis ist, in dem WebSphere MQ 7 installiert ist.
2. Fügen Sie eine Anweisung *using* zum Namensbereich `IBM.XMS.WCF` hinzu. Beispiel: `using IBM.XMS.WCF`
3. Erstellen Sie eine Instanz der Bindungseigenschaften für den Kanal und des Endpunkts, wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen" auf Seite 644 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 zur Verwendung des angepassten WebSphere 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 zum 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

Anweisungen zum Verwenden des Microsoft-Tools "svcutil.exe" zum Generieren eines Clients für einen Service, der für die Verwendung des angepassten WebSphere 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.
- Der WCF-Service muss seine Metadaten zusätzlich zu den Endpunktreferenzen des angepassten WebSphere MQ-Kanals auch über einen HTTP-Port verfügbar machen, um einen Client direkt aus einem aktiven Service zu generieren.
- Der angepasste Kanal muss in den Konfigurationsdaten für 'svcutil' registriert sein.

Informationen zu diesem Vorgang

In den folgenden Schritten wird erläutert, wie Sie einen Client für einen Service erstellen, der für die Verwendung des angepassten WebSphere MQ-Kanals konfiguriert ist, der jedoch seine Metadaten zur Laufzeit auch 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 Details aus der Konfigurationsdatei `svcutil.exe` aus dem Stammverzeichnis der Installation in der Konfigurationsdatei des aktiven "svcutil"-Tools hinzu. Dabei handelt es sich in der Regel um `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`. So erkennt "svcutil" den angepassten WebSphere MQ-Kanal.
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 Clientprojekt von Microsoft Visual Studio.

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 652

WCF-Client-Proxy und Anwendungskonfigurationsdateien mithilfe des Tools 'svcutil' mit WSDL generieren

In diesem Abschnitt finden Sie Anweisungen zum Generieren von WCF-Clients aus WSDL, wenn die Metadaten des Service nicht verfügbar sind.

Wenn die Metadaten des Service zum Generieren eines Clients aus den Metadaten eines aktiven Service nicht direkt abgerufen werden können, können die Clientdateien stattdessen mithilfe des Tools 'svcutil' aus WSDL generiert werden. Die folgenden Änderungen an der WSDL müssen vorgenommen werden, um anzugeben, dass der angepasste WebSphere MQ-Kanal verwendet werden soll:

1. Fügen Sie die folgenden Namensbereichsdefinitionen und Richtlinieninformationen hinzu:

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utili
ty-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. Ändern Sie den Abschnitt zu den Bindungen, damit auf den Abschnitt mit der neuen Richtlinie verwiesen wird, und entfernen Sie alle `transport`-Definitionen aus dem zugrundeliegenden Bindungselement:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Führen Sie das Tool 'svcutil' aus einer Eingabeaufforderung auf; Beispiel:

```
svcutil /language:C# /r:MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
        /config:app.config MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service\soap.ser
ver.stockQuoteAxis_Wmq.wsdl
```

Dabei ist *MQ_INSTALLATION_PATH* das Installationsverzeichnis von WebSphere MQ.

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 642 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. `IBM.XMS.WCF.dll` befindet sich im Verzeichnis *WMQInstallDir*`\bin` , wobei *WMQInstallDir* das Verzeichnis ist, in dem WebSphere MQ 7 installiert ist.
2. Fügen Sie eine Anweisung *using* zum Namensbereich `IBM.XMS.WCF` hinzu. Beispiel: `using IBM.XMS.WCF`
3. Erstellen Sie eine Instanz des Bindungselements und des Endpunkt für den Kanal, wie unter „Angepassten WCF-Kanal durch das Bereitstellen von Bindungs- und Endpunktinformationen programmgesteuert erstellen" auf Seite 644 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.Nojn
di");
```

2. Erstellen Sie den Client-Proxy, wie in der folgenden Abbildung gezeigt wird. Dabei sind *Bindung* und *Endpunktadresse* die in Schritt „1” auf Seite 653 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

Die Beispiele für Windows Communication Foundation (WCF) stellen einige einfache Beispiele zur Verwendung des angepassten WebSphere 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 WebSphere MQ-Kanal zum Starten eines WCF-Service (WCF - Windows Communication Foundation) über einen 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 651).

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 WebSphere MQ. Weitere Informationen zum Formatieren des JMS-Endpunkt-URI finden Sie in *WebSphere MQ Transport for SOAP* in der WebSphere MQ-Produktdokumentation. Wenn Sie die Beispiellösung und die Beispielquelle ändern müssen, benötigen Sie eine integrierte Entwicklungsumgebung (IDE) wie beispielsweise Microsoft Visual Studio 8 oder höher.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager mit der Bezeichnung *QM1*.
2. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleQ*.
3. Starten Sie den Service, damit der Listener auf Nachrichten warten kann: Führen Sie die Datei *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von IBM WebSphere MQ.

4. Client einmal ausführen: Führen Sie die Datei *MQ_INSTALLATION_PATH*\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe aus, wobei *MQ_INSTALLATION_PATH* das Installationsverzeichnis für IBM WebSphere MQist.

   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 WebSphere MQ-Kanal zum Starten eines WCF-Service (WCF - Windows Communication Foundation) über einen WCF-Client in Form eines Anforderungs-/Antwortkanals 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 651).

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\WCF\requestreply\client\app.config und in der 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 WebSphere MQ. Weitere Informationen zum Formatieren des JMS-Endpunkt-URI finden Sie in *WebSphere MQ Transport for SOAP* in der WebSphere MQ-Produktdokumentation. Wenn Sie die Beispiellösung und die Beispielquelle ändern müssen, benötigen Sie eine integrierte Entwicklungsumgebung (IDE) wie beispielsweise Microsoft Visual Studio 8 oder höher.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager mit der Bezeichnung *QM1*.
2. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleQ*.
3. Erstellen Sie ein Warteschlangenziel mit der Bezeichnung *SampleReplyQ*.
4. Starten Sie den Service, damit der Listener auf Nachrichten warten kann: Führen Sie die Datei *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe aus. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von WebSphere 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 WebSphere 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 Verbindung zwischen WCF-Client und einem von WebSphere MQ gehosteten .NET-Service

Beispielclientanwendungen und Beispielservice-Proxy-Anwendungen werden sowohl für .NET als auch für 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 die Hosting-Umgebung für den .NET SOAP over JMS-Service ordnungsgemäß in WebSphere MQ installiert und konfiguriert sein und der Zugriff darauf muss über einen lokalen Warteschlangenmanager möglich sein. Informationen zur Installation und Konfiguration der Umgebung finden Sie unter „WebSphere MQ-Webtransport für SOAP installieren” auf Seite 1016

Wenn die Hosting-Umgebung für den .NET SOAP over JMS-Service ordnungsgemäß in WebSphere MQ installiert und konfiguriert ist und der Zugriff darauf über einen lokalen Warteschlangenmanager möglich ist, müssen weitere Konfigurationsschritte ausgeführt werden.

1. Setzen Sie die Umgebungsvariable WMQSOAP_HOME auf das WebSphere MQ-Installationsverzeichnis, beispielsweise auf `C:\Program Files\IBM\WebSphere MQ`.
2. Stellen Sie sicher, dass der Java-Compiler `javac` verfügbar ist und in PATH angegeben ist.
3. Kopieren Sie die Datei `axis.jar` aus dem Verzeichnis `prereqs/axis` der Installations-CD von WebSphere in das Produktionsverzeichnis von WebSphere MQ . Beispiel: `C:\Program Files\IBM\WebSphere MQ\java\lib\soap`
4. Fügen Sie dem Pfad folgende Information hinzu: *MQ_INSTALLATION_PATH*`\Java\lib`. Dabei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem WebSphere MQ installiert ist, z. B. `C:\Program Files\IBM\WebSphere MQ`.
5. Vergewissern Sie sich, 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 WebSphere MQ installiert ist. Beispiel: `C:\Program Files\IBM\WebSphere MQ`. Die Position von .NET kann angegeben werden. Beispiel: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Wenn die bisherigen Schritte abgeschlossen sind, testen Sie den Service und führen ihn aus:

1. Navigieren Sie zu Ihrem SOAP over JMS-Arbeitsverzeichnis.
2. Geben Sie einen der folgenden Befehle zum Ausführen des Funktionstests ein und starten Sie den Listener für den Service:
   - Für .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem WebSphere MQ installiert ist.
   - Für AXIS:*MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` . Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem WebSphere 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 SOAP over JMS-Verzeichnis.
2. Löschen Sie den Warteschlangenmanager.

Informationen zu diesem Vorgang

In diesem Beispiel wird eine Verbindung von einem WCF-Client zu dem .NET SOAP over JMS-Beispielservice gezeigt, der in WebSphere MQ in Form eines Einwegkanals bereitgestellt wird. Der Service implementiert ein einfaches Beispiel für Börsennotierung, 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 652 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 von WebSphere MQ. Weitere Informationen zum Formatieren des JMS-Endpunkt-URI finden Sie in *WebSphere MQ Transport for SOAP* in der WebSphere 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 WebSphere 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 'WCF client to an Axis Java service hosted by WebSphere MQ '

Für Java und .NET werden Beispielclientanwendungen und Beispielservice-Proxy-Anwendungen 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 die Hosting-Umgebung für den .NET SOAP over JMS-Service ordnungsgemäß in WebSphere MQ installiert und konfiguriert sein und der Zugriff darauf muss über einen lokalen Warteschlangenmanager möglich sein. Informationen zur Installation und Konfiguration der Umgebung finden Sie unter „WebSphere MQ-Webtransport für SOAP installieren” auf Seite 1016

Wenn die Hosting-Umgebung für den .NET SOAP over JMS-Service ordnungsgemäß in WebSphere MQ installiert und konfiguriert ist und der Zugriff darauf über einen lokalen Warteschlangenmanager möglich ist, müssen weitere Konfigurationsschritte ausgeführt werden.

1. Setzen Sie die Umgebungsvariable WMQSOAP_HOME auf das WebSphere MQ-Installationsverzeichnis, beispielsweise auf `C:\Program Files\IBM\WebSphere MQ`.
2. Stellen Sie sicher, dass der Java-Compiler `javac` verfügbar ist und in PATH angegeben ist.
3. Kopieren Sie die Datei `axis.jar` vom Verzeichnis `prereqs/axis` der WebSphere-Installations-CD ins WebSphere MQ-Installationsverzeichnis.
4. Fügen Sie dem Pfad folgende Information hinzu: *MQ_INSTALLATION_PATH*`\Java\lib`. Dabei steht *MQ_INSTALLATION_PATH* für das Verzeichnis, in dem WebSphere MQ installiert ist, z. B. `C:\Program Files\IBM\WebSphere MQ`.
5. Vergewissern Sie sich, 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 WebSphere MQ installiert ist. Beispiel: `C:\Program Files\IBM\WebSphere MQ`. Die Position

von .NET kann angegeben werden. Beispiel: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Wenn die bisherigen Schritte abgeschlossen sind, testen Sie den Service und führen ihn aus:

1. Navigieren Sie zu Ihrem SOAP over JMS-Arbeitsverzeichnis.
2. Geben Sie einen der folgenden Befehle zum Ausführen des Funktionstests ein und starten Sie den Listener für den Service:
   - Für .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , wobei *MQ_INSTALLATION_PATH* das Verzeichnis ist, in dem WebSphere MQ installiert ist.
   - Für AXIS:*MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` . Dabei ist *MQ_INSTALLATION_PATH* das Verzeichnis, in dem WebSphere 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 SOAP over JMS-Verzeichnis.
2. Löschen Sie den Warteschlangenmanager.

Informationen zu diesem Vorgang

Das Beispiel veranschaulicht eine Verbindung von einem WCF-Client zum Axis Java SOAP over JMS-Beispielservice, der in WebSphere MQ über einen unidirektionalen Kanal bereitgestellt wird. Der Service implementiert ein einfaches Beispiel für Börsennotierung, 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 652 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 WebSphere 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 WebSphere 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 667
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 'WCF client to Java service hosted by WebSphere Application Server'

Beispielclientanwendungen und Beispiele für Service-Proxy-Anwendungen werden für WebSphere Application Server (WAS) 6 bereitgestellt. Es wird auch ein Anfrage-/Antwort-Service bereitgestellt.

Vorbereitende Schritte

Dieses Beispiel erfordert, dass die folgende WebSphere MQ-Konfiguration verwendet wird:

Tabelle 77. Erforderliche WebSphere 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 V6 ordnungsgemäß installiert und konfiguriert sein. WebSphere Application Server V6 verwendet eine Verbindung im Bindungsmodus, um standardmäßig die Verbindung zu WebSphere MQ herzustellen. Deshalb muss WebSphere Application Server V6 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 von WebSphere Application Server:
   a. Ein JMS-Warteschlangenziel mit dem Namen `HelloWorld`
      - Legen Sie `jms/HelloWorld` als JNDI-Namen fest.
      - Legen Sie `HelloWorld` als Warteschlangenname fest.
   b. Eine JMS-Warteschlangenverbindungs-Factory mit dem Namen `HelloWorldQCF`
      - Legen Sie `jms/HelloWorldQCF` als JNDI-Namen fest.
      - Legen Sie `QM1` als Warteschlangenmanagernamen fest.
   c. Eine JMS-Warteschlangenverbindungs-Factory mit dem Namen `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 dem Namen `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 Web-Service-Anwendung `HelloWorldEJBEAR.ear` folgendermaßen auf Ihrem WebSphere Application Server:
   a. Klicken Sie auf **Anwendungen** > **Neue Anwendung** > **Neue Unternehmensanwendung**.
   b. Navigieren Sie zu *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`. Dabei steht *MQ_INSTALLATION_PATH* für das Installationsverzeichnis von WebSphere 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 WebSphere 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 WebSphere MQ V7 integrierten WCF-Beispielen bereitgestellt wird. In diesem Beispiel wird ein Anforderungs-/Antwortkanal verwendet. Nachrichten werden zwischen WCF und dem WebSphere Application Server mit WebSphere MQ-Warteschlangen übertragen. Der Service implementiert die Methode `HelloWorld(...)` , die eine Zeichenfolge verwendet und eine Begrüßung an den Client zurückgibt.

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 651).

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 WebSphere MQ. Weitere Informationen zur Formatierung des JMS-Endpunkt-URI finden Sie unter URI-Syntax und -Parameter für Web-Service-Implementierung .

Der Service und der Client basieren auf dem Service und dem Client, die im IBM Developer-Artikel *Building a JMS Web service using SOAP over JMS and WebSphere Studio*beschrieben werden. Weitere Informationen zur Entwicklung von SOAP-over-JMS-Web-Services, die mit dem angepassten WCF-Kanal von WebSphere MQ kompatibel sind, finden Sie unter 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 WebSphere 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 WebSphere MQ

Mit dem WebSphere MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des WebSphere 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 WebSphere MQ-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” auf Seite 661.

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. 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 Code für die Fehlernachricht der aktuellen Ausnahme des angepassten Kanals.
- IBM.XMS.ErrorCode: Die Fehlernachricht der ersten XMS-Ausnahme im Stack.
- IBM.WMQ.ReasonCode: Der zugrunde liegende WebSphere MQ-Ursachencode.
- IBM.WMQ.CompletionCode: Der zugrunde liegende WebSphere MQ-Beendigungscode.

WCF-Tracekonfiguration und Namen von Tracedateien

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. Zur Vereinfachung der Traceaktivierung verwenden die .NET 3-Tracequelle und die XMS .NET-Trace-Stacks eine einzelne Schnittstelle.

Für den angepassten WCF-Kanal stehen zwei verschiedene Traceverfahren zur Verfügung. Die beiden Traceverfahren können unabhängig voneinander oder zusammen 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

Für die Konfiguration des XMS .NET-Trace müssen Sie einen Codeabschnitt zum Abschnitt <system.diagnostics><sources> in der Datei app.config 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 78. 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 Tracestufeninformationen von Microsoftfinden Sie in Ihrer WCF-Dokumentation oder auf der Webseite Microsoft WCF Tracing: https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

Tabelle 78. Konfigurationsvariablen für den WCF-Trace (Forts.)	
Variable	**Beschreibung**

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"`
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.

Tabelle 78. Konfigurationsvariablen für den WCF-Trace (Forts.)	
Variable	**Beschreibung**

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 665.

Diese Tabelle und die gezeigten Werte sind davon abhängig, ob die zuvor gezeigten Codeteile der Datei `app.config` bereits hinzugefügt wurden.

Tabelle 79. 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" switchVa lue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFi leNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
XMS-Trace ist aktiviert. WCF-Tracequelle ist inaktiviert.	Der Wert `switchValue` ist auf `Off` gesetzt und `xmsTraceSpecification` wurde angegeben.	`<source name="IBM.XMS.WCF" switchVa lue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFi leNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Tabelle 79. Kombinationen zur Aktivierung des WCF-Trace (Forts.)

Tracetyp	Geänderter Wert	Beispiel
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" switchVa lue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFile Size="2000000" xmsTraceFileNumber="4" xmsTraceFor mat="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" switchVa lue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFile Size="2000000" xmsTraceFileNumber="4" xmsTraceFor mat="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: `wcfxms_trace_`*`pid`*`.log`. Dabei ist *pid* die Prozess-ID.

Die Traceausgabedatei wird standardmäßig im aktuellen Arbeitsverzeichnis erstellt, aber dieses Ziel kann bei Bedarf neu definiert werden.

First Failure Support Technology (FFST) für WCF XMS

Mit dem WebSphere MQ-Trace können Sie ausführliche Informationen zu den Funktionen der verschiedenen Teile des WebSphere MQ-Codes erfassen. XMS FFST verfügt über eigene Konfigurations- und Ausgabedateien für den angepassten WCF-Kanal.

XMS FFST -Tracedateien werden traditionell unter Verwendung des Basisnamens und des Prozess-ID-Formats `xmsffdc`*`pid_date`*`.txt`benannt, wobei *pid* die Prozess-ID und *date* die Uhrzeit und das Datum ist.

Da XMS FFST -Tracedateien parallel zu Dateien des angepassten WCF-Kanals XMS FFST erstellt werden können, haben die Ausgabedateien des angepassten WCF-Kanals XMS FFST das folgende Format, um Unklarheiten zu vermeiden: `wcfffdc`*`pid_date`*`.txt`. Dabei ist *pid* die Prozess-ID und *date* 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 WebSphere MQ-Kanal für Metadaten der WCF-Version kann auf eine von drei Arten abgerufen werden:

- Mit dem WebSphere MQ-Dienstprogramm 'dspmqver'. Informationen zur Verwendung von 'dspmqver' finden Sie unter dspmqver.
- Im Eigenschaftendialog des Windows -Explorers: Klicken Sie im Windows -Explorer 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 FFST-Headerinformationen finden Sie unter „First Failure Support Technology (FFST) für WCF XMS” auf Seite 665.

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, Collection<ServiceEndpoint> 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.

WCF hat die Konvention, dass der zurückgegebene Wert im folgenden Format benannt werden soll: *methodName*`Result` . Dabei ist *methodName* 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.

C++ verwenden

WebSphere MQ bietet C++-Klassen, die WebSphere MQ-Objekten entsprechen, sowie einige zusätzliche Klassen, die den Array-Datentypen entsprechen. Die Lösung beinhaltet zahlreiche Funktionen, die über die MQI nicht zur Verfügung stehen.

Ab WebSphere MQ Version 7.0 werden Verbesserungen der WebSphere MQ-Programmierschnittstellen nicht mehr auf die C++-Klassen angewendet.

WebSphere MQ C++ stellt die folgenden Funktionen bereit:

- Automatische Initialisierung von WebSphere 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 WebSphere 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.

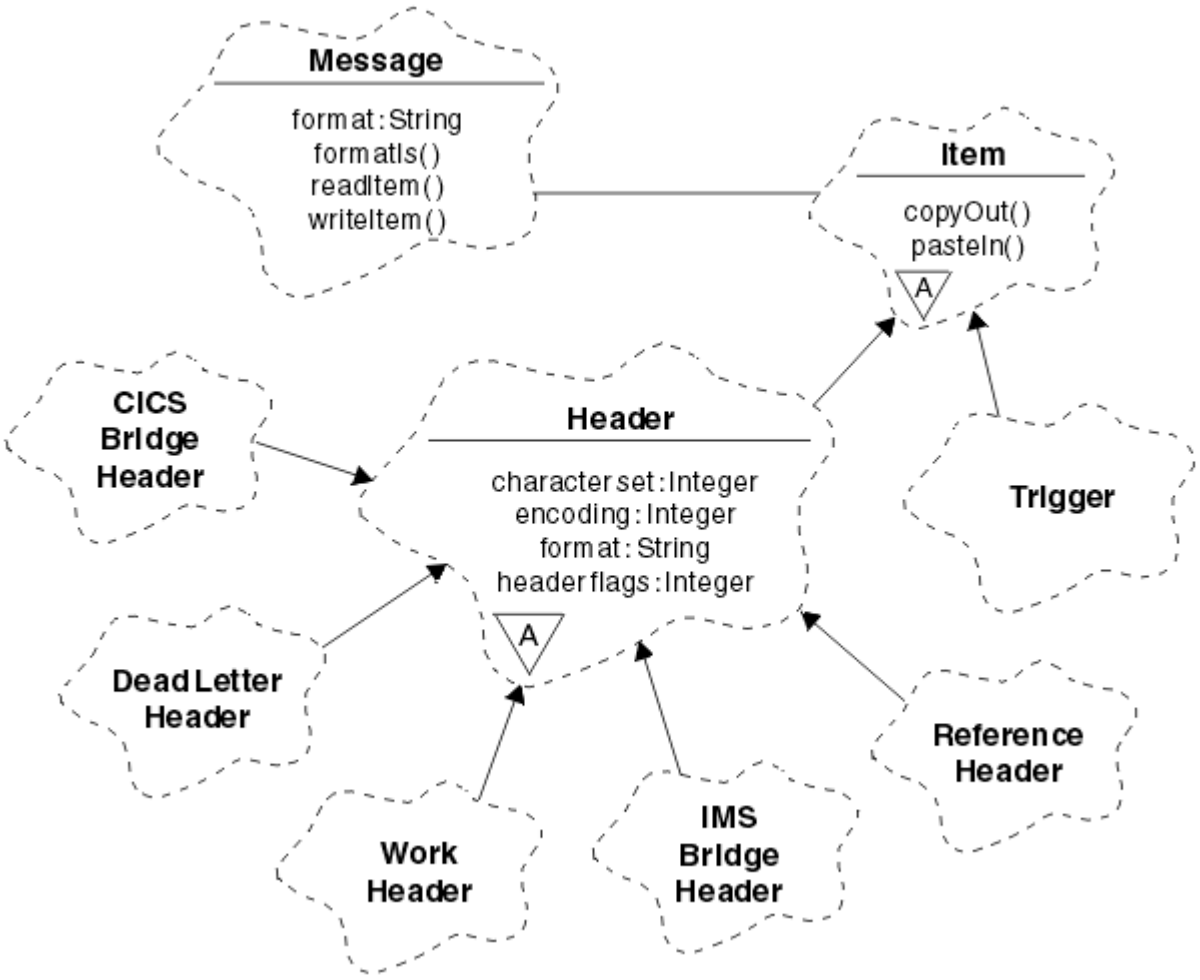


Abbildung 120. WebSphere MQ C++-Klassen (Elementverarbeitung)

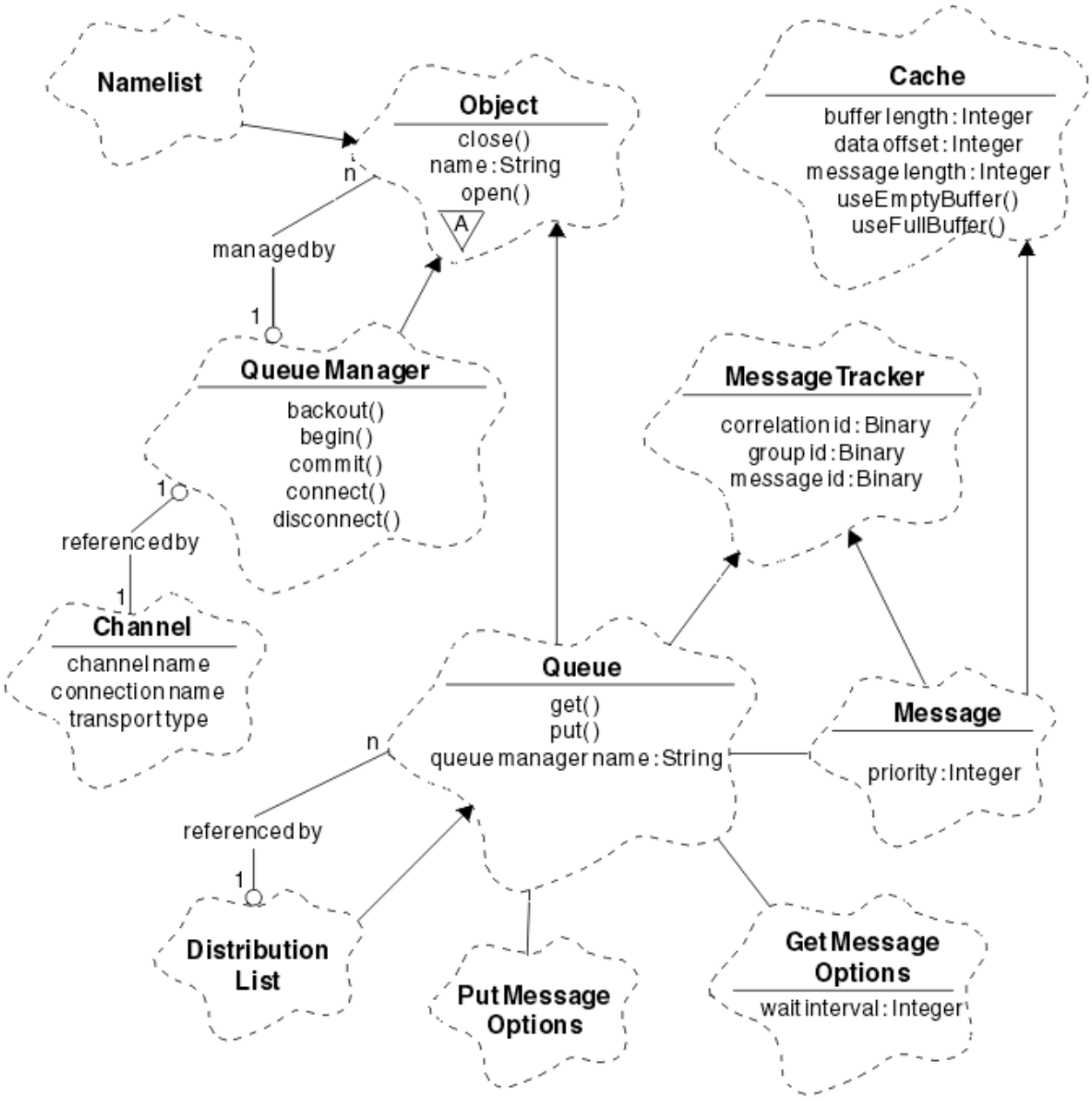


Abbildung 121. WebSphere MQ C++-Klassen (Warteschlangenmanagement)

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 121 auf Seite 670) und der Klassen zur Elementverarbeitung (siehe Abbildung 120 auf Seite 669) 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 681), indem zwischengespeicherte Pufferfunktionen bereitgestellt werden. Sie können Puffer mit fester Länge für Benutzerdaten bereitstellen und den Puffer vielfach verwenden. Das im Puffer enthaltene Datenvolumen kann von einer Verwendung zur nächsten unterschiedlich sein. Das System kann alternativ einen Puffer mit flexibler Länge bereitstellen und verwalten. Die Größe des Puffers (der verfügbare Umfang für den Empfang von Nachrichten) und der tatsächlich verwendete Umfang (die Anzahl der Byte für die Übertragung oder die Anzahl der tatsächlich empfangenen Byte) werden zu wichtigen Überlegungen.

Zugehörige Konzepte

Technische Übersicht

„Beispielprogramme in C++” auf Seite 671
In vier bereitgestellten Beispielprogrammen wird das Abrufen und Einreihen von Nachrichten veranschaulicht.

„Überlegungen zur Programmiersprache C++” auf Seite 675
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 680
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.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„Anwendungen entwickeln” auf Seite 7
IBM WebSphere MQ bietet mehrere Möglichkeiten, mit denen Sie Anwendungen entwickeln können, um Nachrichten zur Unterstützung Ihrer Geschäftsprozesse zu senden und zu empfangen. Darüber hinaus können Sie Anwendungen für das Management Ihrer Warteschlangenmanager und zugehörigen Ressourcen entwickeln.

Zugehörige Verweise

„WebSphere MQ-Programme in C++ erstellen” auf Seite 686
Die URL unterstützter Compiler wird zusammen mit den zu verwendenden Befehlen zum Kompilieren, Verbinden und Ausführen von C++-Programmen und Beispielen zu WebSphere MQ-Plattformen aufgelistet.

Querverweis für C++ und MQI

WebSphere 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 80 auf Seite 672 gezeigten Verzeichnissen.

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

Tabelle 80. Position der Beispielprogramme

Umgebung	**Verzeichnis mit Quelle**	**Verzeichnis mit erstellten Programme**
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ah (siehe Hinweis „2" auf Seite 672)
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 672)

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 687.
3. Programme, die mithilfe von Microsoft Visual Studio erstellt wurden, befinden sich in *MQ_INSTAL-LATION_PATH*\tools\cplus\samples\bin\vn. Weitere Informationen zu diesen Compilern finden Sie unter „C++-Programme unter Windows erstellen" auf Seite 693.

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.

Bei allen Plattformen außer z/OS

Wenn Sie eine Serververbindung mit WebSphere MQ verwenden, führen Sie einen der folgenden Schritte aus:

- 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 mit WebSphere MQ verwenden, führen Sie einen der folgenden Schritte aus:

- Richten Sie die Umgebungsvariable MQSERVER ein (weitere Informationen finden Sie unter MQSERVER) und führen Sie **imqwrldc** aus, oder
- Führen Sie **imqwrldc** aus, indem Sie die Parameter **queue-name**, **queue-manager-name**und **channel-definition**übergeben, wobei ein typisches **channel-definition** SYSTEM.DEF.SVRCONN/TCP/*Hostname*(1414)

Beispielcode

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // WebSphere 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.

Bei 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.

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 WebSphere MQ-Anwendungsprogrammen in der Programmiersprache C++ bereitgestellt.

Diese Headerdateien werden in der folgenden Tabelle zusammengefasst.

Tabelle 81. 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 WebSphere 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.

In diesem Codebeispiel werden die Methoden und Parameter**ImqBoolean ImqQueue**::**get**(**ImqMessage &** *msg*) verwendet.

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 WebSphere 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 WebSphere MQ C++-Klassen und -Methoden sind unabhängig von der WebSphere 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 WebSphere MQ erkannt, aber nicht von den Programmiersprachenbindungen von C++. WebSphere 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 WebSphere 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 );
```

Nachricht an die IMS-Bridge in C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht an die IMS-Bridge.

Nachrichten, die an die WebSphere MQ-IMS-Bridge 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 an die CICS-Bridge in C++ schreiben

Beispielprogrammcode für das Schreiben einer Nachricht an die CICS-Bridge.

An WebSphere MQ for z/OS über die CICS -Bridge gesendete Nachrichten erfordern einen speziellen 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 an eine Warteschlange, die vom z/OS Workload Manager verwaltet wird.

Nachrichten, die an WebSphere MQ for z/OS gesendet werden und für eine Warteschlange bestimmt sind, die vom z/OS Workload Manager verwaltet 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 );
```

WebSphere MQ-Programme in C++ erstellen

Die URL unterstützter Compiler wird zusammen mit den zu verwendenden Befehlen zum Kompilieren, Verbinden und Ausführen von C++-Programmen und Beispielen zu WebSphere MQ-Plattformen aufgelistet.

Die Compiler für jede unterstützte Plattform und Version von WebSphere MQ werden auf der Systemanforderungenseite von WebSphere MQ unter Systemvoraussetzungen für IBM WebSphere MQ aufgelistet.

Der Befehl, mit dem Sie das WebSphere MQ C++-Programm kompilieren und verbinden müssen, hängt von der Installation und den Anforderungen ab. In den folgenden Beispielen sind typische Befehle zum Kompilieren und Verbinden für einige der Compiler zu sehen, die die Standardinstallation von WebSphere MQ bei einer Vielzahl von Plattformen verwenden.

C++-Programme unter AIX erstellen

WebSphere MQ C++-Programme unter AIX mit dem Compiler der XL C Enterprise Edition erstellen.

Client

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

32-Bit Unthread-Anwendung

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32-Bit Thread-Anwendung

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64-Bit Unthread-Anwendung

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64-Bit Thread-Anwendung

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

Server

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

32-Bit Unthread-Anwendung

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32-Bit Thread-Anwendung

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64-Bit Unthread-Anwendung

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64-Bit Thread-Anwendung

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

C++-Programme unter HP-UX erstellen

WebSphere MQ C++-Programme unter HP-UX mithilfe des aC++- oder aCC-Compilers erstellen.

Unter HP-UX Itanium unterstützt WebSphere MQ nur die Standardlaufzeit. Verwenden Sie den aCC-Compiler.

- libimqi23bh.sl stellt die WebSphere MQ C++-Klassen für die Standardlaufzeit 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 WebSphere MQ installiert ist.

Client: IA64 (IPF)

32-Bit Unthread-Anwendung

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

C++-Programme unter Linux erstellen

Erstellen Sie WebSphere MQ C + + -Programme unter Linux mit dem GNU g + + -Compiler.

System p

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

Client: System p

32-Bit Unthread-Anwendung

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

System z

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

Client: System z

32-Bit Unthread-Anwendung

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64  -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Server: System z

32-Bit Unthread-Anwendung

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 WebSphere MQ installiert ist.

Client: System x (32-Bit)

32-Bit Unthread-Anwendung

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATHlib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

C++-Programme unter Solaris erstellen

WebSphere MQ C++-Programme unter Solaris mithilfe des Sun ONE-Compilers erstellen.

SPARC

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

Client: SPARC

32-Bit-Anwendung

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 WebSphere MQ installiert ist.

Client: x86-64

32-Bit-Anwendung

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
```

```
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

C++-Programme unter Windows erstellen

WebSphere MQ C++-Programme unter Windows mithilfe des Microsoft Visual Studio C++-Compilers erstellen.

Bibliotheksdateien (.lib) und DLL-Dateien zur Nutzung mit 32-Bit-Anwendungen sind in *MQ_INSTALLATION_PATH*/Tools/Lib installiert, Dateien zur Verwendung mit 64-Bit-Anwendungen in *MQ_INSTALLATION_PATH*/Tools/Lib64.*MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere 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
```

WebSphere MQ Classes for Java verwenden

Mit WebSphere MQ Classes for Java können Sie WebSphere MQ in einer Java-Umgebung verwenden. Eine Java-Anwendung kann entweder WebSphere MQ Classes for Java oder WebSphere MQ Classes for JMS verwenden, um auf WebSphere MQ -Ressourcen zuzugreifen.

WebSphere MQ Classes for Java ermöglicht einer Java-Anwendung Folgendes:

- Eine Verbindung zu WebSphere MQ als WebSphere MQ-Client herstellen
- Eine direkte Verbindung zu einem WebSphere MQ-Warteschlangenmanager herstellen

WebSphere MQ -Klassen für Java kapseln Message Queue Interface (MQI), die native WebSphere MQ -API.

WebSphere MQ -Klassen für Java verwenden ein ähnliches Objektmodell wie die C + +-und .NET-Schnittstellen für WebSphere MQ.

Warum sollte ich WebSphere MQ Classes for Java verwenden?

Wenn die folgenden Punkte in Ihrer Installation von Bedeutung sind, ziehen Sie die Verwendung von WebSphere MQ -Klassen für Java in Betracht:

- WebSphere MQ -Klassen für Java kapseln Message Queue Interface (MQI), die native WebSphere MQ -API.
  - Wenn Sie mit der Verwendung der MQI in prozeduralen Sprachen vertraut sind, können Sie dieses Wissen in die Java-Umgebung übertragen.
  - Sie können alle Funktionen von WebSphere MQ nutzen, nicht nur die über JMS verfügbaren Funktionen.
- WebSphere MQ -Klassen für Java verwenden ein ähnliches Objektmodell wie die C + +-und .NET-Schnittstellen für WebSphere MQ. Wenn Sie mit diesen Schnittstellen vertraut sind, können Sie dieses Wissen in die Java-Umgebung übertragen.

Anmerkung: Die automatische Clientverbindungswiederholung wird von WebSphere MQ Classes for Java nicht unterstützt.

Erste Schritte mit WebSphere MQ Classes for Java

Diese Themensammlung enthält eine Übersicht über WebSphere MQ -Klassen für Java und ihre Verwendung.

Was sind WebSphere MQ -Klassen für Java?

WebSphere MQ Classes for Java ermöglicht die Verwendung von WebSphere MQ in einer Java-Umgebung.

WebSphere MQ Classes for Java ermöglicht einer Java-Anwendung Folgendes:

- Eine Verbindung zu WebSphere MQ als WebSphere MQ-Client herstellen
- Eine direkte Verbindung zu einem WebSphere MQ-Warteschlangenmanager herstellen

WebSphere MQ -Klassen für Java kapseln Message Queue Interface (MQI), die native WebSphere MQ -API.

WebSphere MQ -Klassen für Java verwenden ein ähnliches Objektmodell wie die C + +-und .NET-Schnittstellen für WebSphere MQ.

Warum sollte ich WebSphere MQ Classes for Java verwenden?

Eine Java-Anwendung kann entweder WebSphere MQ Classes for Java oder WebSphere MQ Classes for JMS verwenden, um auf WebSphere MQ -Ressourcen zuzugreifen. Die Verwendung von WebSphere MQ -Klassen für Java bietet eine Reihe von Vorteilen.

Wenn die folgenden Punkte in Ihrer Installation von Bedeutung sind, ziehen Sie die Verwendung von WebSphere MQ -Klassen für Java in Betracht:

- WebSphere MQ -Klassen für Java kapseln Message Queue Interface (MQI), die native WebSphere MQ -API.
  - Wenn Sie mit der Verwendung der MQI in prozeduralen Sprachen vertraut sind, können Sie dieses Wissen in die Java-Umgebung übertragen.
  - Sie können alle Funktionen von WebSphere MQ nutzen, nicht nur die über JMS verfügbaren Funktionen.
- WebSphere MQ -Klassen für Java verwenden ein ähnliches Objektmodell wie die C + +-und .NET-Schnittstellen für WebSphere MQ. Wenn Sie mit diesen Schnittstellen vertraut sind, können Sie dieses Wissen in die Java-Umgebung übertragen.

Verbindungsoptionen für WebSphere MQ -Klassen für Java

WebSphere MQ Classes for Java kann eine Verbindung im Client-oder Bindungsmodus herstellen.

Programmierbare Optionen ermöglichen WebSphere MQ Classes for Java, eine Verbindung zu WebSphere MQ auf eine der folgenden Arten herzustellen:

- Als WebSphere MQ-Client mithilfe von Transmission Control Protocol/Internet Protocol (TCP/IP)
- Im Bindungsmodus: direkte Verbindung zu WebSphere MQ über JNI (Java Native Interface)

Clients können nicht unter z/OS ausgeführt werden, aber Clients auf anderen Plattformen können eine Verbindung zu einem WebSphere MQ for z/OS-Warteschlangenmanager herstellen, wenn die Client Attachment-Komponente installiert ist.

In den folgenden Abschnitten werden die Verbindungsoptionen des Clientmodus und Bindungsmodus genauer beschrieben.

Clientverbindung

Um eine Verbindung zu einem Warteschlangenmanager im Client-Modus herzustellen, kann eine WebSphere MQ Classes for Java-Anwendung auf demselben System ausgeführt werden, auf dem der Warte-

schlangenmanager ausgeführt wird, oder auf einem anderen System. In jedem Fall stellt WebSphere MQ Classes for Java über TCP/IP eine Verbindung zum Warteschlangenmanager her.

Eine Anwendung, die die WebSphere MQ-Klassen für Java verwendet, kann eine Verbindung zu jedem unterstützten Warteschlangenmanager mithilfe des Clientmodus herstellen.

Weitere Informationen zum Erstellen von Anwendungen, die Verbindungen im Clientmodus verwenden, finden Sie im Abschnitt „WebSphere MQ Classes for Java-Verbindungsmodi” auf Seite 709.

Bindings- Verbindung

Im Bindungsmodus verwendet WebSphere MQ Classes for Java die Java Native Interface (JNI), um direkt in die vorhandene Warteschlangenmanager-API aufzurufen, anstatt über ein Netz zu kommunizieren. In den meisten Umgebungen bietet die Verbindung im Bindungsmodus eine bessere Leistung für WebSphere MQ -Klassen für Java-Anwendungen als die Verbindung im Clientmodus, da die Kosten für die TCP/IP-Kommunikation vermieden werden.

Anwendungen, die WebSphere MQ-Klassen für 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 zur Ausführung der WebSphere MQ -Klassen für die Java-Anwendung verwendet wird, muss so konfiguriert werden, dass sie die WebSphere MQ -Klassen für Java-Bibliotheken lädt. Weitere Informationen finden Sie im Abschnitt WebSphere MQ -Klassen für Java-Bibliotheken .

Weitere Informationen zum Erstellen von Anwendungen, die Verbindungen im Bindungsmodus verwenden, finden Sie im Abschnitt „WebSphere MQ Classes for Java-Verbindungsmodi” auf Seite 709.

Voraussetzungen für WebSphere MQ Classes for Java

Zur Verwendung der WebSphere MQ -Klassen für Java benötigen Sie bestimmte andere Softwareprodukte.

Die neuesten Informationen zu den Voraussetzungen für WebSphere MQ -Klassen für Java finden Sie in der Readme-Datei zu WebSphere MQ .

Zur Entwicklung von WebSphere MQ -Klassen für Java-Anwendungen benötigen Sie ein Java Development Kit (JDK). Details zu den JDKs, die von Ihrem Betriebssystem unterstützt werden, erfahren Sie auf der Systemanforderungenseite von WebSphere MQ unter Systemvoraussetzungen für IBM WebSphere MQ.

Zum Ausführen von WebSphere MQ -Klassen für Java-Anwendungen benötigen Sie die folgenden Softwarekomponenten:

- Einen WebSphere MQ-Warteschlangenmanager für Anwendungen, die eine Verbindung zu einem Warteschlangenmanager herstellen
- Eine Java Runtime Environment (JRE) für jedes System, auf dem Anwendungen ausgeführt werden. Eine geeignete JRE wird mit WebSphere MQ bereitgestellt.

Wenn Sie SSL-Verbindungen zur Verwendung von Verschlüsselungsmodulen benötigen, die gemäß FIPS 140-2 zertifiziert sind, benötigen Sie den IBM Java JSSE FIPS-Provider (IBMJSSEFIPS). Jedes IBM JDK und jede JRE ab Version 1.4.2 enthält IBMJSSEFIPS.

Sie können Adressen von Internet Protocol Version 6 (IPv6) in Ihren WebSphere MQ -Klassen für Java-Anwendungen verwenden, wenn IPv6 von Ihrer JVM (Java Virtual Machine) und der TCP/IP-Implementierung auf Ihrem Betriebssystem unterstützt wird.

Installation und Konfiguration von WebSphere MQ Classes for Java

In diesem Kapitel werden die Verzeichnisse und Dateien beschrieben, die bei der Installation von WebSphere MQ Classes for Java erstellt werden. Außerdem erfahren Sie, wie Sie WebSphere MQ Classes for Java nach der Installation konfigurieren.

Was ist für WebSphere MQ Classes for Java installiert?

Die neueste Version von WebSphere MQ Classes for Java wird zusammen mit WebSphere MQinstalliert. Es ist möglicherweise erforderlich, hierfür Standardinstallationsoptionen aufzuheben.

Weitere Informationen zum Installieren von WebSphere MQ finden Sie unter:

WebSphere MQ-Server installieren
IBM WebSphere MQ-Client installieren

WebSphere MQ -Klassen für Java sind in den JAR-Dateien com.ibm.mq.jarund com.ibm.mq.jmqi.jarenthalten.

Unterstützung für standardmäßige Nachrichtenheader, wie 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.

Installieren und Aktualisieren der JAR-Dateien von WebSphere MQ-Klassen für Java

Die einzige unterstützte Möglichkeit, die JAR-Dateien von WebSphere MQ-Klassen für Java auf einem System bereitzustellen, besteht darin, entweder WebSphere MQ zu installieren, das WebSphere MQ-Client SupportPac zu installieren oder ein Softwareverwaltungstool wie Apache Maven zu verwenden. Weitere Informationen finden Sie unter „IBM WebSphere MQ classes for Java und Software-Management-Tools” auf Seite 705.

Verschieben oder kopieren Sie die JAR-Dateien von WebSphere MQ-Klassen für Java nicht von anderen Systemen, es sei denn, Sie verwenden ein Softwareverwaltungstool.

- Fixpacks können nicht auf eine "Installation" angewendet werden, in der JAR-Dateien von einem anderen System kopiert wurden. Zudem ist es deutlich schwieriger, sicherzustellen, dass alle JAR-Dateien gleich aktuell sind und kompatible Versionen haben.
- Das Kopieren von JAR-Dateien von WebSphere MQ-Klassen für JMS von einem System auf ein anderes kann auch dazu führen, dass auf einem System mehrere Kopien der Dateien vorhanden sind, was Probleme bei der Wartung des Codes und beim Debugging verursachen kann.

Schließen Sie die JAR-Dateien von WebSphere MQ-Klassen für Java nicht in Anwendungsarchive ein.

- Aktualisierungen für WebSphere MQ-Klassen für Java können nicht mithilfe eines WebSphere MQ-Fixpacks angewendet werden.
- Es ist schwierig für den IBM Support, die Version von WebSphere MQ-Klassen für Java, die von der Anwendung verwendet wird, zu bestimmen.
- Wenn mehrere Anwendungen, die in derselben Java Runtime Environment ausgeführt werden, verschiedene Versionen der WebSphere MQ-Klassen für Java enthalten, kann dies Probleme verursachen, da mehrere Versionen der WebSphere MQ-Klassen für Java gleichzeitig in die Java Runtime Environment geladen werden.
- Wenn eine Anwendung den Bindungstransport verwendet, um eine Verbindung zu einem Warteschlangenmanager herzustellen, muss für wichtige Upgrades des Warteschlangenmanagers auch die Anwendung aktualisiert werden, damit sie die entsprechende Version von WebSphere MQ-Klassen für Java enthält.

  Wenn ein Warteschlangenmanager z. B. auf WebSphere MQ Version 7.1 aktualisiert wird, müssen auch alle Anwendungen, die mithilfe des Bindungstransports eine Verbindung zum Warteschlangenmanager herstellen, aktualisiert werden, damit sie Version 7.1 der WebSphere MQ-Klassen für Java enthalten.

Die folgende Java-Bibliothek wird mit WebSphere MQ -Klassen für Java verteilt:

- connector.jar (Version 1.0)

Die Beispielanwendung mit der Bezeichnung 'Postcard' ist in der JAR-Datei com.ibm.mq.postcard.jar enthalten.

Das Javadoc-Tool wurde zum Generieren der HTML-Seiten verwendet, die die Spezifikationen der WebSphere MQ -Klassen für Java und WebSphere MQ -Klassen für JMS-APIs enthalten. Die HTML-Seiten befinden sich im Unterverzeichnis 'doc' des WebSphere MQ-Klassen für JMS-Installationsverzeichnisses. Auf Systemen unter UNIX, Linux und Windows enthält das Unterverzeichnis 'doc' die einzelnen HTML-Seiten.

Wenn die Installation abgeschlossen ist, sind die Dateien und Beispiele in den Verzeichnissen installiert, die in „Installationsverzeichnisse für WebSphere MQ -Klassen für Java” auf Seite 697 angezeigt sind.

Nach der Installation müssen Sie auf allen Plattformen außer Windows Ihre Umgebungsvariablen aktualisieren, wie in „Für WebSphere MQ -Klassen für Java relevante Umgebungsvariablen” auf Seite 697 beschrieben.

Installationsverzeichnisse für WebSphere MQ -Klassen für Java

WebSphere MQ Classes for Java-Dateien werden je nach Plattform an verschiedenen Positionen installiert.

Tabelle 82 auf Seite 697 zeigt, wo die Dateien von WebSphere MQ Classes for Java installiert sind.

Tabelle 82. WebSphere MQ Classes for Java-Installationsverzeichnisse

Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/java/lib
HP-UX, Linuxund Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.	

Es werden einige Beispielanwendungen, wie z. B. die Installationsprüfprogramme, mit WebSphere MQ bereitgestellt.Tabelle 83 auf Seite 697 zeigt, wo die Beispielanwendungen installiert sind. Die Beispiele für WebSphere MQ Classes for Java befinden sich in einem Unterverzeichnis mit dem Namen `wmqjava`. Die PCF-Beispiele sind in einem Unterverzeichnis namens `pcf`.

Tabelle 83. Beispielverzeichnisse

Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
HP-UX, Linuxund Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.	

Für WebSphere MQ -Klassen für Java relevante Umgebungsvariablen

Wenn Sie WebSphere MQ -Klassen für Java-Anwendungen ausführen möchten, müssen ihre Klassenpfade die Verzeichnisse WebSphere MQ -Klassen für Java und Beispiele enthalten.

Damit WebSphere MQ -Klassen für Java-Anwendungen ausgeführt werden können, muss ihr Klassenpfad das entsprechende Verzeichnis für WebSphere MQ -Klassen für Java enthalten. Um die Beispielanwendungen auszuführen, muss der Klassenpfad ebenfalls die entsprechenden Beispielverzeichnisse aufweisen. Diese Informationen können im Java-Aufrufbefehl oder in der Umgebungsvariablen CLASSPATH angegeben werden.

Tabelle 84 auf Seite 698 zeigt die entsprechende CLASSPATH-Einstellung, die auf jeder Plattform verwendet werden kann, um WebSphere MQ -Klassen für Java-Anwendungen auszuführen, einschließlich der Beispielanwendungen.

Tabelle 84. CLASSPATH-Einstellung für die Ausführung von WebSphere MQ -Klassen für Java-Anwendungen, einschließlich der WebSphere MQ -Klassen für Java-Beispielanwendungen	
Plattform	**CLASSPATH-Einstellung**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
HP-UX, Linuxund Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere 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 IBM WebSphere MQ Advanced Message Security installiert ist.

Die mit WebSphere MQ -Klassen für Java 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 WebSphere MQ -Klassen für Java installiert sind (siehe WebSphere MQ -Klassen für Java-Installationsverzeichnisse).

MQ_JAVA_LIB_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem die Bibliotheken der WebSphere MQ -Klassen für Java gespeichert sind (siehe Die Position der Bibliotheken der WebSphere MQ -Klassen für Java für jede Plattform). Einige Scripts, die mit WebSphere MQ Classes for Java bereitgestellt werden, wie z. B. IVTRun, verwenden diese Umgebungsvariable.

Unter Windows werden alle Umgebungsvariablen bei der Installation automatisch festgelegt. Auf allen anderen Plattformen müssen Sie sie selbst einstellen.Auf einem UNIX-System können Sie die Umgebungsvariablen mit dem Script **setjmsenv** (bei Verwendung einer 32-Bit-JVM) bzw. **setjmsenv64** (bei Verwendung einer 64-Bit-JVM) festlegen. Unter AIX, HP-UX, Linuxund Solaris befinden sich diese Scripts im Verzeichnis *MQ_INSTALLATION_PATH*/java/bin .

IBM WebSphere MQ -Klassen für Java-Bibliotheken

Die Position der IBM WebSphere MQ -Klassen für Java-Bibliotheken variiert je nach Plattform. Geben Sie diesen Standort an, wenn Sie eine Anwendung starten.

Zur Angabe der Position der JNI-Bibliotheken (JNI = Java Native Interface) starten Sie Ihre Anwendung mit einem **java** -Befehl im folgenden Format:

```
java -Djava.library.path=library_path application_name
```

Dabei ist *Bibliothekspfad* der Pfad zu den Bibliotheken von WebSphere MQ Classes for Java, die die JNI-Bibliotheken enthalten. Tabelle 85 auf Seite 699 zeigt die Position der Bibliotheken von WebSphere MQ Classes for Java für jede Plattform.

Tabelle 85. Die Position der Bibliotheken von WebSphere MQ Classes for Java für jede Plattform	
Plattform	**Verzeichnis mit den Bibliotheken der WebSphere MQ -Klassen für Java**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
HP-UX Linux (POWER, x86-64 und zSeries s390x -Plattformen) Solaris (x86-64- und SPARC-Plattformen)	*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)
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.	

Anmerkung:

1. Verwenden Sie unter AIX, HP-UX, Linux (Power-Plattform) oder Solaris entweder die 32-Bit-Bibliotheken oder die 64-Bit-Bibliotheken. Verwenden Sie die 64-Bit-Bibliotheken nur, wenn Sie Ihre Anwendung in einer 64-Bit-Java Virtual Machine (JVM) auf einer 64-Bit-Plattform ausführen. Verwenden Sie ansonsten die 32-Bit-Bibliotheken.
2. Unter Windowskönnen Sie die Umgebungsvariable PATH verwenden, um die Position der Bibliotheken der WebSphere MQ -Klassen für Java anzugeben, anstatt sie im Befehl **java** anzugeben.
3. Wenn Sie WebSphere MQ Classes for Java im Bindungsmodus unter IBM iverwenden möchten, stellen Sie sicher, dass die Bibliothek QMQMJAVA in Ihrer Bibliotheksliste enthalten ist.

Zugehörige Tasks

WebSphere MQ Classes for Java verwenden

Unterstützung für OSGi mit IBM WebSphere MQ classes for Java

OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Ein OSGi-Bundle wird als Teil der IBM WebSphere 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 dynamische und skalierbare Weise.

IBM WebSphere MQ classes for Java enthält das folgende OSGi-Bundle.

com.ibm.mq.osgi.java_< Versionsnummer > .jar
Die JAR-Dateien, um Anwendungen die Verwendung von IBM WebSphere MQ classes for Java zu ermöglichen.

Dabei ist < Versionsnummer > die Versionsnummer von WebSphere MQ , die installiert wurde.

Das Bundle wird im Unterverzeichnis `java/lib/OSGi` der IBM WebSphere MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert.

Es werden ebenfalls neun weitere Bundles im Unterverzeichnis `java/lib/OSGi` der IBM WebSphere MQ-Installation oder im Ordner `java\lib\OSGi` unter Windows installiert. Diese Bundles sind Teil der IBM WebSphere MQ classes for JMS und dürfen nicht in eine OSGi-Laufzeitumgebung geladen werden, in der das IBM WebSphere MQ classes for Java-Bundle geladen wurde. Wenn das OSGi-Bundle für die IBM WebSphere MQ classes for Java in eine OSGi-Laufzeitumgebung geladen wird, in die auch die Bundles für die IBM WebSphere MQ classes for JMS geladen wurden, treten Fehler wie

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

auf, wenn Anwendungen ausgeführt werden, die das Bundle für die IBM WebSphere MQ classes for Java oder die Bundles für die IBM WebSphere MQ classes for JMS verwenden.

Das OSGi-Bundle für IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java-Bundle Bescheid wissen, aber das IBM WebSphere MQ classes for Java-Bundle weiß über kein anderes Benutzer-Bundle Bescheid. Deshalb kann das Klassenladeprogramm, das in einem IBM WebSphere MQ classes for Java-Bundle verwendet wird, keine Kanalexitklasse laden, die sich in einem Benutzerbundle befindet.

Weitere Informationen über OSGi finden Sie auf der Website der Open Service Gateway-Initiative.

Die Konfigurationsdatei für die IBM WebSphere MQ classes for Java

Eine IBM WebSphere MQ classes for Java -Konfigurationsdatei gibt Eigenschaften an, die für die Konfiguration des IBM WebSphere MQ classes for Javaverwendet werden.

Das Format einer IBM WebSphere MQ classes for Java -Konfigurationsdatei entspricht dem einer standardmäßigen Java -Eigenschaftendatei.

V 7.5.0.9 Ab IBM WebSphere MQ Version 7.5.0Fixpack 9wird eine Beispielkonfigurationsdatei namens `mqjava.config` im Unterverzeichnis `bin` des Installationsverzeichnisses von IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java angibt. Unterstützt werden URLs der folgenden Typen: `http`, `file`, `ftp` und `jar`.

Hier ein Beispiel für einen **java**-Befehl:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

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

Eine Konfigurationsdatei für IBM WebSphere MQ classes for Java kann mit allen unterstützten Transportprotokollen zwischen einer Anwendung und einem Warteschlangenmanager oder Broker verwendet werden.

In einer IBM WebSphere MQ classes for Java-Konfigurationsdatei angegebene Eigenschaften überschreiben

In einer IBM WebSphere MQ MQI client-Konfigurationsdatei können auch Eigenschaften angegeben werden, die zur Konfiguration der IBM WebSphere MQ classes for Java verwendet werden. Eigenschaften, die in einer IBM WebSphere 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 WebSphere MQ MQI client überschreiben, indem Sie es als Eigenschaft in einer Konfigurationsdatei für IBM WebSphere MQ classes for Java angeben. Wenn Sie ein Attribut in einer Konfigurationsdatei für den IBM WebSphere MQ MQI client überschreiben möchten, verwenden Sie in der Konfigurationsdatei für IBM WebSphere 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 WebSphere MQ MQI client, die das Attribut enthält.

propName
Der Name des Attributs, so wie in der Konfigurationsdatei für den IBM WebSphere MQ MQI client angegeben.

propValue
Der Wert der Eigenschaft, der den Wert des Attributs überschreibt, das in der Konfigurationsdatei für den IBM WebSphere MQ MQI client angegeben ist.

Alternativ können Sie ein Attribut in einer IBM WebSphere 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 WebSphere MQ MQI client -Konfigurationsdatei sind für IBM WebSphere MQ classes for Javarelevant. 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. Im Abschnitt „Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for Java verwenden” auf Seite 714 finden Sie Details zur Verwendung der CCDT bei den IBM WebSphere MQ classes for Java.

Tabelle 86. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut

Zeilengruppe	Attribut
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	Standardpfad für Exits
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	ExitsDefaultPath64
Zeilengruppe 'ClientExitPath' der Clientkonfigurationsdatei	JavaExitsClasspath
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	MaximumSize

Tabelle 86. Welche Zeilengruppe der Clientkonfigurationsdatei enthält welches Attribut (Forts.)	
Zeilengruppe	**Attribut**
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	PurgeTime
Zeilengruppe 'MessageBuffer' der Clientkonfigurationsdatei	UpdatePercentage
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntRcvBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	Connect_Timeout
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	KeepAlive

Weitere Informationen zur Konfiguration des IBM WebSphere MQ MQI clients finden Sie im Abschnitt Client mithilfe einer Konfigurationsdatei konfigurieren.

Zugehörige Tasks

Trace für Anwendungen erstellen, die die IBM WebSphere MQ classes for Java verwenden

Zeilengruppe für den Java-Standardumgebungstrace

Mithilfe der Zeilengruppe ' Java Standard Environment Trace Settings' können Sie die Tracefunktion von IBM WebSphere MQ classes for Java konfigurieren.

com.ibm.msg.client.commonservices.trace.outputName = `traceOutputName`

`traceOutputName` ist das Verzeichnis und der Dateiname, an die die Traceausgabe gesendet wird.

Der Standardname der Tracedatei hängt von der Version der IBM WebSphere MQ classes for Java ab, die von einer Anwendung verwendet wird:

- Bei IBM WebSphere MQ classes for Java für Version 7.5.0, Fix Pack 8 oder früher ist *traceOutputName* standardmäßig eine Datei mit dem Namen `mqjms_%PID%.trc` im aktuellen Arbeitsverzeichnis.
- V 7.5.0.9 Ab IBM WebSphere MQ classes for Java ab Version 7.5.0, Fix Pack 9ist *traceOutputName* standardmäßig eine Datei mit dem Namen `mqjava_%PID%.trc` im aktuellen Arbeitsverzeichnis.

Dabei ist *%PID%* die aktuelle Prozess-ID. Wenn eine Prozess-ID nicht verfügbar ist, wird eine Zufallszahl generiert und mit dem Buchstaben `f` als Präfix versehen. Um die Prozess-ID in einen von Ihnen angegebenen Dateinamen einzuschließen, verwenden Sie die Zeichenfolge `%PID%` .

Wenn Sie ein anderes Verzeichnis angeben, muss dieses bereits vorhanden sein und Sie müssen über Schreibzugriff auf dieses Verzeichnis verfügen. Wenn Sie nicht über Schreibzugriff verfügen, wird die Traceausgabe in die Datei `System.err` geschrieben.

com.ibm.msg.client.commonservices.trace.include = `includeList`

`includeList` ist eine Liste der Pakete und Klassen, für die ein Trace erstellt wird, oder die Sonderwerte `ALL` oder `NONE`.

Trennen Sie Paket-oder Klassennamen durch ein Semikolon (`;`). **`includeList`** nimmt standardmäßig den Wert `ALL`an und zeichnet alle Pakete und Klassen in IBM WebSphere MQ classes for Javaauf.

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`.

Trennen Sie Paket-oder Klassennamen durch ein Semikolon (;). **excludeList** nimmt standardmäßig den Wert NONEan und schließt daher keine Pakete und Klassen in IBM WebSphere MQ classes for Java 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 = maxArrayBytes
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 die in *maxBytesFeldgruppe* angegebene Byteanzahl 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.

Wird der über **maxTraceBytes** angegebene Grenzwert für die aktuelle Traceausgabe erreicht, wird die Datei geschlossen. Die weitere Traceausgabe wird in die nächste Traceausgabedatei in der Folge geschrieben. Die einzelnen Traceausgabedateien sind durch ein numerisches Suffix am Ende des Dateinamens eindeutig gekennzeichnet. Die aktuelle bzw. zuletzt erstellte Traceausgabedatei hat das Suffix .trc.0, die davor erstellte Traceausgabedatei hat das Suffix .trc.1 usw. Bis zum Erreichen des Höchstwerts wird dieses Nummerierungsmuster auch auf die älteren Traceausgabedateien angewandt.

Der Standardwert von **traceCycles** ist 1. Wenn **traceCycles** auf 1gesetzt ist und die aktuelle Traceausgabedatei die 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.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 der Dateiname für die Traceausgabe jedoch in **traceOutputName** angegeben ist, wird stattdessen dieser Name verwendet und es wird kein Suffix auf die Datei 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. Ein komprimierter Trace beeinträchtigt die Leistung der IBM WebSphere MQ classes for Java in geringerem Maße als ein nicht komprimierter Trace.

Wenn **maxTraceBytes** und **traceCycles** festgelegt sind, werden mehrere komprimierte Tracedateien anstelle mehrerer Flachdateien erstellt.

Bei einer nicht gesteuerten Beendigung der IBM WebSphere MQ classes for Java ist die komprimierte Tracedatei unter Umständen ungültig. Daher darf ein komprimierter Trace nur verwendet werden, wenn die IBM WebSphere MQ classes for Java gesteuert beendet werden. Die Tracekomprimierung sollte nur zum Einsatz kommen, wenn die Probleme, die untersucht werden sollen, keine unerwartete Beendigung der JVM selbst verursachen. Daher sollte die Tracekomprimierung nicht verwendet werden, wenn Probleme untersucht werden, die zu einem Systemstopp (`System.Halt()`) oder zu einer abnormalen, nicht gesteuerten JVM-Beendigung führen könnten.

com.ibm.msg.client.commonservices.trace.level = traceLevel
traceLevel gibt eine Filterstufe für den Trace an Die definierten Tracestufen lauten wie folgt:

Tabelle 87. Traceumfang der einzelnen Tracestufen

Wert	Traceumfang
0	Trace ist inaktiviert
1	Ausnahmen
3	Ausnahmen Warnungen
6	Ausnahmen Warnungen Informationstracepunkte
8	Ausnahmen Warnungen Informationstracepunkte Methodenein- und austritt
9	Ausnahmen Warnungen Informationstracepunkte Methodenein- und austritt Daten, die zwischen den IBM WebSphere MQ classes for Java und einem Warteschlangenmanager ausgetauscht werden.

Anmerkung: Verwenden Sie immer den Wert 9 , sofern Sie nicht vom IBM Support dazu aufgefordert werden.

IBM WebSphere MQ classes for Java und Software-Management-Tools

Mit den IBM WebSphere 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 WebSphere MQ classes for Java setzen sich aus einer Reihe von JAR-Dateien zusammen. Wenn Sie Java-Sprachanwendungen mithilfe dieser API entwickeln, ist eine Installation von IBM WebSphere MQ Server, IBM WebSphere MQ Client oder IBM WebSphere MQ Client SupportPac auf der Maschine erforderlich, auf der die Anwendung entwickelt wird.

Wenn Sie ein Softwareverwaltungstool verwenden und die JAR-Dateien, die die IBM WebSphere 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 WebSphere MQ-Release oder -Fixpack enthalten.
- Es ist Ihre Aufgabe, das Repository mit allen vom IBM Support zur Verfügung gestellten Wartungsreleases zu aktualisieren.

Bei IBM WebSphere MQ Version 7.5 müssen die folgenden JAR-Dateien im Repository installiert werden:

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- connector.jar

Konfiguration für Anwendungen der IBM WebSphere MQ nach der Installation

Nach der Installation von IBM WebSphere MQ können Sie die Installation für die Ausführung eigener Anwendungen konfigurieren.

Denken Sie daran, sich in der Readme-Datei von IBM WebSphere MQ zu informieren, um zu einem späteren Zeitpunkt weitere oder spezifischere Informationen zu Ihrer Umgebung zu erhalten.

Bevor Sie versuchen, eine IBM WebSphere MQ -Klassen für Java -Anwendung im Bindungsmodus auszuführen, stellen Sie sicher, dass Sie IBM WebSphere MQ wie im Abschnitt Konfiguration beschrieben konfiguriert haben.

Warteschlangenmanager für das Akzeptieren von Clientverbindungen von WebSphere MQ -Klassen für 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 „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

WebSphere MQ -Klassen für Java-Anwendungen unter Java Security Manager ausführen

WebSphere MQ Classes for Java kann mit aktiviertem Java Security Manager ausgeführt werden. Für eine erfolgreiche Ausführung von Anwendungen mit aktiviertem Sicherheitsmanager müssen Sie Ihre JVM (Java Virtual Machine) mit einer geeigneten Richtliniendefinitionsdatei konfigurieren.

Die einfachste Methode besteht darin, die zusammen mit der JRE bereitgestellte Richtliniendatei zu ändern. Auf den meisten Systemen ist diese Datei im Pfad `lib/security/java.policy` (relativ zum JRE-Verzeichnis) gespeichert. Sie können Richtliniendateien mit Ihrem bevorzugten Editor oder dem `policytool`-Programm, das mit Ihrer JRE bereitgestellt wurde, bearbeiten.

Sie müssen die Berechtigung für die Datei com.ibm.mq.jmqi.jar erteilen, damit sie Folgendes ausführen kann:

- Sockets erstellen (im Clientmodus)
- Native Bibliothek laden (im Bindungsmodus)
- Verschiedene Eigenschaften aus der Umgebung lesen

Die Systemeigenschaft os.name muss für die WebSphere MQ -Klassen für Java verfügbar sein, wenn sie unter Java Security Manager ausgeführt wird.

Es folgt ein Beispiel für einen Richtliniendateieintrag, der die erfolgreiche Ausführung von WebSphere MQ -Klassen für Java unter dem Standardsicherheitsmanager ermöglicht:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //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
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase
"file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.commonservices.jar" {
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.mq.commonservices", "read";
  permission java.util.logging.LoggingPermission "control";
  //For access to the trace properties file.
  permission java.io.FilePermission "/tmp/trace.properties", "read";
  //For access to the trace output files.
  permission java.io.FilePermission "/tmp/*", "read,write";
};
```

Anmerkungen:

- *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.
- Dieses Beispiel einer Richtliniendatei ermöglicht es den WebSphere MQ -Klassen für Java, unter dem Sicherheitsmanager ordnungsgemäß zu arbeiten, aber Sie müssen möglicherweise Ihren eigenen Code aktivieren, damit er ordnungsgemäß ausgeführt wird, bevor Ihre Anwendungen funktionieren.
- Damit WebSphere MQ Classes for Java auf die JAR-Dateien einer Anwendung zugreifen kann, fügen Sie der ersten Anweisung `grant` die folgende Berechtigung hinzu:

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- Um diese `grant` -Anweisungen in Ihrer Richtlinienkonfigurationsdatei zu verwenden, müssen Sie möglicherweise die Pfadnamen ändern, je nachdem, wo Sie WebSphere MQ -Klassen für Java installiert haben und wo Sie Ihre Anwendungen speichern.

- Der Beispielcode, der im Lieferumfang von WebSphere MQ Classes for Java enthalten ist, wurde nicht speziell für die Verwendung mit dem Sicherheitsmanager aktiviert. Die IVT-Tests werden jedoch mit dieser Richtliniendatei und dem Standardsicherheitsmanager ausgeführt.

Installation der IBM WebSphere MQ-Klassen für Java überprüfen

Mit IBM WebSphere MQ -Klassen für Javawird ein Installationsprüfprogramm (MQIVP) bereitgestellt. Mithilfe dieses Programms können Sie sämtliche Verbindungsmodi von IBM WebSphere MQ-Klassen für 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 „Beispielprogramme vorbereiten und ausführen” auf Seite 115beschrieben. Die zu verwendende Warteschlange ist SYSTEM.DEFAULT.LOCAL.QUEUE.
2. Wenn Sie das Programm im Clientmodus ausführen möchten, informieren Sie sich auch unter „WebSphere MQ Classes for Java verwenden” auf Seite 693.

   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 „Für WebSphere MQ -Klassen für Java relevante Umgebungsvariablen” auf Seite 697 aktualisiert haben.
4. Wechseln Sie in das Verzeichnis `MQ_INSTALLATION_PATH/mqm/VRM/java/samples/wmqjava`, wobei MQ_INSTALLATION_PATH der Pfad zu Ihrer IBM WebSphere MQ -Installation und `VRM` die Version, das Release und die Modifikationsnummer des Produkts ist. Geben Sie dann bei der Eingabeaufforderung Folgendes ein:

   ```
   java -Djava.library.path=library_path MQIVP
   ```

   Dabei ist *bibliothekspfad* der Pfad zu den IBM WebSphere MQ -Klassen für Java -Bibliotheken (siehe Die WebSphere MQ -Klassen für Java-Bibliotheken).

   Gehen Sie bei der Eingabeaufforderung mit der Markierung (1) folgendermaßen vor:

   - Um eine TCP/IP-Verbindung zu verwenden, geben Sie einen IBM WebSphere 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 WebSphere 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. Wenn Sie die Serververbindung wählen, sehen Sie keine Eingabeaufforderungen mit der Markierung (2).

Probleme in Zusammenhang mit den IBM WebSphere MQ 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 „Installation der IBM WebSphere MQ-Klassen für Java überprüfen” auf Seite 707 beschrieben.

Wenn die Fehler weiterhin auftreten und Sie den IBM Kundendienst kontaktieren müssen, werden Sie möglicherweise dazu aufgefordert, die Tracefunktion zu aktivieren. Gehen Sie hierzu wie im folgenden Beispiel gezeigt vor.

So verfolgen Sie das MQIVP-Programm:

- Erstellen Sie eine Eigenschaftendatei *com.ibm.mq.commonservices* (siehe com.ibm.mq.commonservices).
- 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* ist der Pfad zu den WebSphere MQ -Klassen für Java-Bibliotheken (siehe WebSphere MQ -Klassen für Java-Bibliotheken).

Weitere Informationen zur Verwendung der Tracefunktion finden Sie unter Trace für Anwendungen erstellen, die die IBM WebSphere MQ classes for Java verwenden.

Einführung für Programmierer

Die vorliegende Themensammlung enthält allgemeine Informationen für Programmierer.

Ausführlichere Informationen zum Schreiben von Programmen erhalten Sie unter „ WebSphere MQ -Klassen für Java-Anwendungen schreiben” auf Seite 709.

Die Schnittstelle WebSphere MQ Classes for Java

Die prozedurale WebSphere MQ-Anwendungsprogrammierschnittstelle verwendet Verben, die auf Objekte einwirken. Die Java-Programmierschnittstelle verwendet Objekte, auf die Sie durch Aufrufen von Methoden reagieren.

Die prozedurale WebSphere MQ-Anwendungsprogrammierschnittstelle basiert auf Verben, wie z. B.:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Diese Verben verwenden alle als Parameter eine Kennung für das WebSphere MQ-Objekt, auf dem sie ausgeführt werden sollen. Da Java objektorientiert ist, dreht die Java-Programmierschnittstelle diese Runde. Ihr Programm besteht aus einer Gruppe von WebSphere MQ-Objekten, mit denen Sie arbeiten, indem Sie Methoden zu diesen Objekten aufrufen.

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 durch ein Objekt der Klasse MQQueueManagerdargestellt. 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();
```

WebSphere MQ -Klassen für Java-Anwendungen schreiben

Diese Themensammlung enthält Informationen, die Sie beim Schreiben von Java-Anwendungen für die Interaktion mit WebSphere MQ -Systemen unterstützen.

Wenn Sie WebSphere MQ Classes for Java für den Zugriff auf WebSphere MQ -Warteschlangen verwenden möchten, schreiben Sie Java-Anwendungen, die Aufrufe enthalten, die Nachrichten in WebSphere MQ -Warteschlangen einreihen und daraus abrufen. Details zu einzelnen Klassen finden Sie im Abschnitt WebSphere MQ -Klassen für Java.

Anmerkung: Die automatische Clientverbindungswiederholung wird von WebSphere MQ Classes for Java nicht unterstützt.

WebSphere MQ Classes for Java-Verbindungsmodi

Die Art und Weise, wie Sie für WebSphere MQ -Klassen für Java programmieren, hat einige Abhängigkeiten von den Verbindungsmodi, die verwendet werden sollen.

Wenn Sie Clientverbindungen verwenden, gibt es eine Reihe von Unterschieden zu IBM WebSphere 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.

WebSphere MQ Classes for Java-Clientverbindungen

Wenn WebSphere MQ Classes for Java als Client verwendet wird, entspricht dies dem IBM WebSphere MQ MQI client, weist jedoch eine Reihe von Unterschieden auf.

Wenn Sie die Programmierung für *WebSphere MQ Classes for Java* für die Verwendung als Client verwenden, müssen Sie die folgenden Unterschiede beachten:

- Es wird nur TCP/IP unterstützt.
- Es werden keine WebSphere 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 Standardfehlerziel ist die Java-Konsole.
- Es sind nur die folgenden Attribute in einer WebSphere MQ-Clientkonfigurationsdatei für WebSphere MQ-Klassen für 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	ClntRcvBufSize
Zeilengruppe 'TCP' der Clientkonfigurationsdatei	ClntSndBufSize
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 WebSphere MQ Classes for Java nicht unterstützt.

Bei Verwendung im Clientmodus unterstützt *WebSphere MQ Classes for Java* den MQBEGIN-Aufruf nicht.

Unter „Verbindungsoptionen für WebSphere MQ -Klassen für Java” auf Seite 694 erhalten Sie weitere Informationen zu unterstützten Umgebungen.

WebSphere MQ Classes for Java-Bindungsmodus

Der Bindungsmodus von WebSphere MQ -Klassen für Java unterscheidet sich vom Clientmodus auf drei Arten.

Im Bindungsmodus verwenden WebSphere MQ-Klassen für Java die Java Native Interface (JNI), um die vorhandene Warteschlangenmanager-API direkt aufzurufen, anstatt über ein Netzwerk zu kommunizieren.

Standardmäßig stellen Anwendungen, die WebSphere MQ-Klassen für Java im Bindungsmodus verwenden, eine Verbindung zu einem Warteschlangenmanager mithilfe der *Verbindungsoption* MQCNO_STANDARD_BINDINGS her.

WebSphere MQ-Klassen für Java unterstützen die folgenden *Verbindungsoptionen*:

- 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 222.

Der Bindungsmodus unterstützt auf allen Plattformen außer WebSphere MQ für IBM i und WebSphere MQ für 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.

Unter „Verbindungsoptionen für WebSphere MQ -Klassen für Java” auf Seite 694 erhalten Sie weitere Informationen zu unterstützten Umgebungen.

Zu verwendende Verbindung für WebSphere MQ Classes for Java definieren

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 WebSphere 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 WebSphere MQ -Klassen für Java eine Verbindung zu einem Warteschlangenmanager herstellen und trennen.

Einrichten der WebSphere MQ -Umgebung für WebSphere MQ Classes for Java

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. Sie sind *nicht* relevant, wenn im Bindungsmodus eine Verbindung hergestellt wird. Weitere Informationen erhalten Sie unter: „Verbindungsmodi für WebSphere MQ Classes for JMS” auf Seite 822

Sie können den Kanalnamen, den Hostnamen und die Portnummer auf eine der folgenden zwei Arten festlegen: entweder als Felder in der MQEnvironment-Klasse oder als Eigenschaften des MQQueueManager-Objekts.

Wenn Sie Felder in der MQEnvironment-Klasse festlegen, gelten sie für Ihre gesamte Anwendung, außer sie werden von einer Eigenschaften-Hashtabelle außer Kraft gesetzt. Um den Kanalnamen und Hostnamen in MQEnvironment anzugeben, verwenden Sie folgenden Code:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

Dies entspricht der Einstellung einer **MQSERVER**-Umgebungsvariablen:

```
"java.client.channel/TCP/host.domain.com".
```

Standardmäßig versuchen die Java-Clients eine Verbindung zu einem WebSphere MQ -Listener an Port 1414herzustellen. Um einen anderen Port anzugeben, verwenden Sie folgenden Code:

```
MQEnvironment.port = nnnn;
```

Dabei steht nnnn für die erforderliche Portnummer.

Wenn Sie Eigenschaften an ein Warteschlangenmanagerobjekt bei seiner Erstellung übergeben, gelten sie ausschließlich für diesen Warteschlangenmanager. Erstellen Sie Einträge in einem Hashtabellenobjekt mit Schlüsseln von **hostname**, **channel** und optional **port** sowie mit entsprechenden Werten. Um den Standardport (1414) zu verwenden, können Sie den Eintrag **port** übergehen. Erstellen Sie das MQQueueManager-Objekt, indem Sie einen Konstruktor verwenden, der die Eigenschaften-Hashtabelle akzeptiert.

Verbindung zum Warteschlangenmanager durch Einstellung eines Anwendungsnamens ermitteln

Eine Anwendung kann einen Namen festlegen, der die Verbindung zum Warteschlangenmanager angibt. Dieser Anwendungsname wird durch den Befehl **DISPLAY CONN MQSC/PCF** (mit dem Feld **APPLTAG**) oder in der Anzeige WebSphere MQ Explorer **Application Connections** (mit dem Feld **App name**) angezeigt.

Anwendungsnamen sind auf 28 Zeichen beschränkt; längere Namen werden entsprechend 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.

Anmerkung: Warteschlangenmanager, die auf z/OS-Plattformen ausgeführt werden, unterstützen das Festlegen von Anwendungsnamen nicht.

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.

In einer WebSphere MQ-Clientkonfigurationsdatei angegebene Eigenschaften überschreiben

Eine WebSphere MQ -Clientkonfigurationsdatei kann auch Eigenschaften angeben, die für die Konfiguration von WebSphere MQ Classes for Java verwendet werden. Die in einer WebSphere MQ-Clientkonfigurati-

onsdatei angegebenen Eigenschaften gelten jedoch nur, wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt.

Falls erforderlich, können Sie alle Attribute in einer WebSphere MQ-Konfigurationsdatei auf folgende Arten überschreiben. Die Optionen werden geordnet nach Ausführungspriorität gezeigt.

- Definieren Sie eine Java-Systemeigenschaft für die Konfigurationseigenschaft.
- Legen Sie die Eigenschaft in der MQEnvironment.properties-Map fest.
- Festlegen einer Systemumgebungsvariablen in Java5 und höheren Releases.

Nur die folgenden Attribute in einer Clientkonfigurationsdatei von WebSphere MQ sind für WebSphere 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 WebSphere MQ Classes for Java herstellen

Stellen Sie eine Verbindung zu einem Warteschlangenmanager her, indem Sie eine neue Instanz der MQQueueManager-Klasse erstellen. Sie können die Verbindung zum Warteschlangenmanager trennen, indem Sie die Methode 'disconnect()' aufrufen.

Sie können nun eine Verbindung zu einem Warteschlangenmanager herstellen, indem Sie eine neue Instanz der MQQueueManager-Klasse erstellen:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Sie können die Verbindung zu einem Warteschlangenmanager trennen, indem Sie die Methode 'disconnect()' beim Warteschlangenmanager aufrufen.

```
queueManager.disconnect();
```

Wenn Sie die Trennungsmethode aufrufen, werden alle offenen Warteschlangen und Prozesse geschlossen, auf die Sie über diesen Warteschlangenmanager zugegriffen haben. Es wird jedoch aus programmiertechnischen Gründen empfohlen, die Ressourcen explizit zu schließen, wenn Sie mit ihrer Verwendung fertig sind. Verwenden Sie hierfür die Methode 'close()' bei den relevanten Objekten.

Die Methoden 'commit()' und 'backout()' bei einem Warteschlangenmanager entsprechen den Aufrufen MQCMIT bzw. MQBACK, die bei der prozeduralen Schnittstelle verwendet werden.

Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for Java verwenden

Eine IBM WebSphere MQ classes for Java -Clientanwendung kann Clientverbindungskanaldefinitionen verwenden, die in einer Clientkanaldefinitionstabelle (CCDT) gespeichert sind.

Als Alternative zum Erstellen einer Clientverbindungskanaldefinition durch Festlegen bestimmter Felder und Umgebungseigenschaften in der Klasse `MQEnvironment` oder durch Übergeben an eine `MQQueueManager` in einer Eigenschaftenhashtabelle kann eine IBM WebSphere MQ classes for Java -Clientanwendung Clientverbindungskanaldefinitionen verwenden, die in einer Clientkanaldefinitionstabelle gespeichert sind. Diese Definitionen werden von MQSC-Befehlen (IBM WebSphere MQ) oder PCF-Befehlen (IBM WebSphere MQ) erstellt oder mit dem IBM WebSphere MQ Explorer.

Wenn die Anwendung ein `MQQueueManager` -Objekt erstellt, durchsucht der IBM WebSphere MQ classes for Java -Client die Definitionstabelle für Clientkanäle nach einer geeigneten Clientverbindungskanaldefinition und verwendet die Kanaldefinition zum Starten 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 Clientkanal-Definitionstabelle 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 Clientkanal-Definitionstabelle und befindet sich auf einem anderen System als dem, auf dem die Anwendung ausgeführt wird. Wenn auf die Datei mit dem FTP-Protokoll zugegriffen werden kann, dann kann die Anwendung folgendermaßen ein URL-Objekt erstellen:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Nachdem die Anwendung ein URL-Objekt erstellt hat, kann die Anwendung ein `MQQueueManager` -Objekt mit einem der Konstruktoren erstellen, die ein URL-Objekt als Parameter verwenden. Beispiel:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Diese Anweisung bewirkt, dass der IBM WebSphere MQ classes for Java -Client auf die Clientkanaldefinitionstabelle zugreift, die durch das URL-Objekt chanTab2angegeben ist, die Tabelle nach einer geeigneten Clientverbindungskanaldefinition durchsucht und dann die Kanaldefinition zum Starten eines MQI-Kanals zum Warteschlangenmanager namens MARS verwendet.

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. Wenn ein Kanalname festgelegt ist, löst der IBM WebSphere 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 finden Sie in „Message Queue Interface (MQI) - Übersicht" auf Seite 207.

Wenn Ihre Anwendung Verbindungspooling nutzt, erhalten Sie weitere Informationen unter „Standard-verbindungspool in WebSphere MQ Classes for Java steuern" auf Seite 736.

- Wenn der Client der IBM WebSphere MQ classes for Java eine geeignete Clientverbindungskanaldefinition in der Definitionstabelle für Clientkanäle findet, verwendet er nur die aus dieser Kanaldefinition extrahierten Informationen zum Starten eines MQI-Kanals. Jegliche kanalbezogenen Felder oder Umgebungseigenschaften, die die Anwendung möglicherweise in der `MQEnvironment`-Klasse festgelegt hat, werden ignoriert.

  Beachten Sie inbesondere die folgenden Punkte, wenn Sie Secure Sockets Layer (SSL) verwenden:

  - Ein MQI-Kanal verwendet SSL nur, wenn die aus der Definitionstabelle für Clientkanäle extrahierte Kanaldefinition den Namen einer CipherSpec angibt, die von dem IBM WebSphere MQ classes for Java -Client 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 Client der IBM WebSphere MQ classes for Java verwendet nur diese Informationen für den Zugriff auf LDAP-Server, die CRLs enthalten.
  - Eine Definitionstabelle für Clientkanäle kann auch die Position eines OCSP-Responders (Online Certificate Status Protocol) enthalten. IBM WebSphere MQ classes for Java können die OCSP-Informationen in einer Clientkanaldefinitionstabellendatei nicht verwenden. Allerdings kann OCSP, wie im Abschnitt Online Certificate Protocol verwenden beschrieben, konfiguriert werden.

  Weitere Informationen zur Verwendung von SSL mit einer Clientkanal-Definitionstabelle erhalten Sie unter Angeben, dass ein MQI-Kanal SSL 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 Javafinden Sie unter „Kanalexit in WebSphere MQ Classes for Java erstellen" auf Seite 729. Weitere Informationen zum Schreiben eines Kanalexits in anderen Sprachen finden Sie unter „Nicht in Java geschriebene Kanalexits mit WebSphere MQ -Klassen für Java verwenden" auf Seite 733.

Portbereich für Clientverbindungen mit IBM WebSphere 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, die die IBM WebSphere MQ classes for Java verwendet, den Versuch unternimmt, im Clientmodus eine Verbindung zu einem IBM WebSphere MQ-Warteschlangenmanager herzustellen, lässt eine Firewall unter Umständen nur Verbindungen von bestimmten Ports oder von einem bestimmten Portbereich aus zu. 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 WebSphere 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).

Zugriff auf Warteschlangen, Themen und Prozesse in WebSphere MQ Classes for Java

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.

Nachrichten in WebSphere MQ Classes for Java verarbeiten

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 prozeduralen Schnittstelle, bei der MQPUT und MQGET Byte-Arrays einreihen und abrufen, reiht die Programmiersprache Java Instanzen der MQMessage-Klasse ein und ruft sie 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.

Wenn die neue Nachrichteninstanz erstellt wird, werden alle MQMD-Parameter automatisch auf ihre Standardwerte gesetzt, wie in Anfangswerte und Sprachendeklarationen 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 1132.

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 (mit readUTF()), ist dies die einfachste Methode zum Senden von Zeichenfoleinformationen.

Leistung nicht persistenter Nachrichten in WebSphere MQ -Klassen für 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 zugehörigen Abschnitt.

In WebSphere MQ-Klassen für 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 unter Verwendung von WebSphere MQ-Klassen für Java asynchron 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 WebSphere MQ-Klassen für Java

In WebSphere MQ-Klassen für Java wird das Thema durch die MQTopic-Klasse dargestellt. Mit den Methoden MQTopic.put() können Sie Veröffentlichungen im Thema vornehmen.

Allgemeine Informationen zu WebSphere MQ Publish/Subscribe finden Sie unter Einführung in WebSphere MQ Publish/Subscribe-Messaging .

Behandlung von WebSphere MQ -Nachrichtenheadern mit WebSphere MQ -Klassen für Java

Es werden Java-Klassen bereitgestellt, die verschiedene Typen von Nachrichtenheadern darstellen. Darüber hinaus stehen zwei Helper-Klassen zur Verfügung.

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.

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 WebSphere MQ-Headerklassen bestimmten Nachrichtentypen und -formaten zugeordnet sind. Die MQHeaderRegistry wird mit allen bekannten aktuellen WebSphere MQ-Formaten und -Headertypen sowie mit deren Implementierungsklassen konfiguriert. Sie können zudem Ihre eigenen Headertypen registrieren.

Die folgenden gängigen WebSphere 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 WebSphere Message Broker gehört. Wird auch für das Speichern von Nachrichteneigenschaften verwendet.
- MQCIH - CICS-Bridge
- MQDLH - Header einer nicht zustellbaren Nachricht
- MQIIH - IMS-Header
- 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 mit WebSphere MQ -Klassen für Java drucken

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 mit WebSphere MQ -Klassen für 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 Nachricht für nicht zustellbare Nachrichten mit WebSphere MQ -Klassen für 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 ());
```

MQDLH mithilfe von WebSphere MQ -Klassen für Java aus einer nicht zustellbaren Nachricht 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 mit WebSphere MQ -Klassen für Java drucken

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 mithilfe von WebSphere MQ -Klassen für 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 mithilfe von WebSphere MQ -Klassen für 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 Beispiel 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 mit WebSphere MQ -Klassen für Java lesen und schreiben

In diesen Beispielen werden die Headerklassen zum Parsen und Bearbeiten des WebSphere 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 WebSphere 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)` bereit. 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 WebSphere 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 mit WebSphere MQ -Klassen für Java erstellen

Sie können Java-Klassen für Headertypen erstellen, die nicht mit WebSphere MQ -Klassen für Java bereitgestellt werden.

Um eine Java-Klasse hinzuzufügen, die einen neuen Headertyp darstellt, den Sie auf dieselbe Weise wie jede mit WebSphere MQ Classes for Java bereitgestellte Headerklasse verwenden können, erstellen Sie eine Klasse, 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.
}
```

Handhabung von PCF-Nachrichten mit WebSphere MQ -Klassen für Java

Java-Klassen werden bereitgestellt, um PCF-strukturierte Nachrichten zu erstellen und zu analysieren und um das Senden von PCF-Anforderungen und das Sammeln von PCF-Antworten zu vereinfachen.

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 WebSphere MQ Classes for Java

Funktionsaufrufe zum Verarbeiten von Nachrichtenkennungen haben keine Entsprechung in WebSphere MQ -Klassen für Java. 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 20.

In WebSphere MQ Classes for Java erfolgt der Zugriff auf Nachrichten über die MQMessage-Klasse. Nachrichtenkennungen werden daher in der Java-Umgebung nicht bereitgestellt und es gibt kein Äquivalent zu den WebSphere MQ -Funktionsaufrufen MQCRTMH, MQDLTMH, MQMHBUF und MQBUFMH.

Zur Festlegung der Eigenschaften von Nachrichtenhandles in der prozedurgesteuerten Schnittstelle verwenden Sie den Aufruf MQSETMP. Verwenden Sie in WebSphere MQ Classes for Java die entsprechende Methode der MQMessage-Klasse:

- 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. Verwenden Sie in WebSphere MQ Classes for Java die entsprechende Methode der MQMessage-Klasse:

- 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. Verwenden Sie in WebSphere MQ Classes for Java die Methode deleteProperty der MQMessage-Klasse.

Fehler in WebSphere MQ Classes for Java behandeln

Behandeln Sie Fehler, die aus WebSphere MQ Classes for Java entstehen, mithilfe von Java `try` -und `catch` -Blöcken.

Methoden in der Java-Schnittstelle geben keinen Beendigungscode und Ursachencode zurück. Stattdessen lösen sie eine Ausnahmebedingung aus, wenn Beendigungscode und Ursachencode nach einem WebSphere MQ-Aufruf nicht beide null sind. Dies vereinfacht die Programmlogik, sodass Sie nicht nach jedem WebSphere MQ-Aufruf die Rückgabecodes überprüfen müssen. Sie können entscheiden, an wel-

chen 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 WebSphere MQ -Aufrufursachencodes, die in Java-Ausnahmebedingungen für z/OS zurückgemeldet werden, sind in Ursachencodes für z/OS und Ursachencodes für alle anderen Plattformen dokumentiert.

Ausnahmebedingungen, die ausgelöst werden, während eine Anwendung der WebSphere MQ -Klassen für Java ausgeführt wird, werden auch 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.

Standardmäßig überträgt die MQException-Klasse Ausnahmebedingungen automatisch an System.err, was für gewöhnlich an die Konsole weitergeleitet wird. Wenn Sie nicht möchten, dass Ausnahmebedingungen auf der Konsole angezeigt werden, fügen Sie in Ihrer Anwendung eine Zeile hinzu, um MQException.log=null festzulegen.

Attributwerte in WebSphere 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 accessQueue()-Methode verwenden, um auf ein Warteschlangenobjekt zuzugreifen, wird dieses Objekt *nicht* automatisch für die Inquire- oder 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 nicht die Parameter SelectorCount, IntAttrCount und CharAttrLength, die in MQINQ gefunden werden, 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 Runtime Environment ist inhärent ein Multithread-Prozess. WebSphere MQ Classes for Java ermöglicht die gemeinsame Nutzung eines Warteschlangenmanagerobjekts durch mehrere Threads, stellt jedoch sicher, dass alle Zugriffe auf den Zielwarteschlangenmanager synchronisiert sind.

Multithread-Programme sind in Java schwer zu vermeiden. 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 Runtime Environment ist inhärent ein Multithread-Prozess. 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 WebSphere MQ-Client wäre dies ein Problem, da die gemeinsame Nutzung von Handles durch mehrere Threads beschränkt ist. WebSphere MQ Classes for Java lockert diese Einschränkung, sodass ein Warteschlangenmanagerobjekt (und die zugehörigen Warteschlangen-, Topic-und Prozessobjekte) von mehreren Threads gemeinsam genutzt werden kann.

Die Implementierung von WebSphere MQ Classes for Java stellt sicher, dass für eine bestimmte Verbindung (ObjektinstanzMQQueueManager) der gesamte Zugriff auf den WebSphere MQ -Zielwarteschlangenmanager synchronisiert wird. 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 WebSphere MQ Classes for Java verwenden

Eine Übersicht über die Verwendung von Kanalexits in einer Anwendung mit den WebSphere MQ -Klassen für Java.

In den folgenden Abschnitten wird beschrieben, wie ein Kanalexit in Java geschrieben wird, wie er zugewiesen wird und 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 WebSphere MQ Classes for Java erstellen

Sie können eigene 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 keinen Java-Kanalexit außerhalb von WebSphere MQ -Klassen für Java verwenden, z. B. wenn Sie eine in C geschriebene Clientanwendung verwenden.

Jede definierte SSL-Verschlüsselung für eine Verbindung 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 WebSphere MQ interpretieren soll.

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 WebSphere MQ 7.0 wurden diese Exits mithilfe der Schnittstellen MQSendExit, MQReceiveExit und MQSecurityExit implementiert, wie im folgenden Beispiel dargestellt. 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 WebSphere MQ classes for Java zuordnen

Sie können einen Kanalexit unter Verwendung von IBM WebSphere MQ classes for Java zuweisen.

Es gibt keine direkte Entsprechung für den IBM WebSphere MQ-Kanal in IBM WebSphere MQ classes for Java. Kanalexits werden einem MQQueueManager zugewiesen. Nachdem sie zum Beispiel eine Klasse definiert hat, die die WMQSecurityExit-Schnittstelle implementiert, kann eine Anwendung den Sicherheitsexit auf eine von vier Arten verwenden:

- Durch die Zuweisung einer Instanz der Klasse zum Feld MQEnvironment.channelSecurityExit, bevor ein MQQueueManager-Objekt erstellt wird
- Durch das Setzen des Feldes MQEnvironment.channelSecurityExit auf eine Zeichenfolge, die die Sicherheitsexitklasse darstellt, bevor ein MQQueueManager-Objekt erstellt wird
- Durch die Erstellung eines Schlüssel/Wert-Paars mit dem Schlüssel CMQC.SECURITY_EXIT_PROPERTY in der Hashtabelle mit Eigenschaften, die an MQQueueManager übergeben wird
- Durch die Verwendung einer Definitionstabelle für den Clientkanal (CCDT)

Jeder Exit, der durch das Setzen des Feldes MQEnvironment.channelSecurityExit auf eine Zeichenfolge, durch die Erstellung eines Schlüssel/Wert-Paars in der Hashtabelle mit Eigenschaften oder mithilfe einer CCDT zugewiesen wird, muss mit einem Standardkonstruktor geschrieben werden. Ein Exit, der als Instanz einer Klasse zugewiesen wird, benötigt keinen Standardkonstruktor (je nach Anwendung).

Eine Anwendung kann auf ähnliche Weise einen Sende- oder Empfangsexit verwenden. Im folgenden Codefragment sehen Sie beispielsweise, wie die Sicherheits-, Sende- und Empfangsexits, die in der Klasse MyMQExits implementiert sind, mit MQEnvironment verwendet werden:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Falls mehrere Methoden für die Zuweisung eines Kanalexits verwendet werden, gilt folgende Ausführungspriorität:

1. Wenn die URL einer CCDT an den MQQueueManager übergeben wird, bestimmen die Inhalte der CCDT die Kanalexits, die verwendet werden sollen. Alle vorhandenen Exitdefinitionen in MQEnvironment oder in der Hashtabelle mit Eigenschaften werden ignoriert.
2. Wenn keine CCDT-URL übergeben wird, werden die Exitdefinitionen aus MQEnvironment und der Hashtabelle zusammengeführt.
   - Wenn der gleiche Exittyp sowohl in MQEnvironment als auch in der Hashtabelle definiert ist, wird die Definition in der Hashtabelle verwendet.
   - Falls funktional entsprechende alte und neue Exittypen angegeben werden (beispielsweise das sendExit-Feld, das nur in IBM WebSphere MQ-Versionen vor Version 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 WebSphere 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 88 auf Seite 732 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 88 auf Seite 732 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 88. Verzeichnis für Kanalexitprogramme

Plattform	**Directory**
AIX, HP-UX, Linuxund 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 WebSphere MQ-Datendateien gewählt haben. Das Standardverzeichnis ist `C:\Program Files\IBM\WebSphere MQ`.	

Daten an Kanalexits in WebSphere 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 WebSphere MQ interpretieren soll.

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 Clientkanäle mit IBM WebSphere MQ classes for Java verwenden” auf Seite 714.

Nicht in Java geschriebene Kanalexits mit WebSphere MQ -Klassen für Java verwenden

Verwendung von Kanalexitprogrammen, die in C von einer Java-Anwendung geschrieben wurden

In WebSphere MQ Version 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 Programmiersprache geschrieben wurde.

Geben Sie den Namen des Exitprogramms im Format `library(function)` an und stellen Sie sicher, dass die Position des Exitprogramms in der Pfadumgebungsvariablen enthalten ist.

Informationen zum Schreiben eines Kanalexits in der Programmiersprache C finden Sie unter „Kanalexitprogramme für Messaging-Kanäle” auf Seite 422.

Externe Exitklassen verwenden

In Versionen von WebSphere MQ vor Version 7.0wurden drei Klassen bereitgestellt, damit Sie Kanalexits verwenden können, 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 zu verwenden, der nicht in Java geschrieben ist, musste eine Anwendung zuerst ein MQExternalSecurity-Exitobjekt 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 88 auf Seite 732 genannten Verzeichnis gespeichert.

Eine Folge von Kanalsende-oder Kanalempfangsexits in WebSphere MQ Classes for Java verwenden

Eine Anwendung der WebSphere MQ -Klassen für Java kann eine Folge von Kanalsende-oder -empfangsexits 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 Schnittstelle MQSendExit implementiert (für einen in Java geschriebenen Sendeexit).
- Eine Instanz der MQExternalSend-Exitklasse (für einen Sendeexit, der nicht in Java geschrieben ist).
- Eine Instanz der MQSendExitChain-Klasse
- Eine Instanz der Zeichenfolgeklasse

Eine Liste kann keine andere Liste enthalten.

Die Anwendung kann auf ähnliche Weise eine Folge von Empfangsexits verwenden.

Wenn eine Zeichenfolge verwendet wird, muss sie aus einer oder mehreren durch Kommas getrennten Exitdefinitionen bestehen, von denen jede den Namen einer Java-Klasse oder ein C-Programm im Format `library(function)` 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. Wenn beispielsweise ein Java-Exit und ein C-Exit verkettet sind, hat das Vorhandensein des Java-Exits keine Auswirkungen auf den C-Exit.

Exitkettenklassen verwenden

In WebSphere MQ-Versionen vor Version 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. Die Verwendung der Schnittstellen von WebSphere MQ Classes for Java bedeutet, dass Ihre Anwendung immer noch eine Abhängigkeit von `com.ibm.mq.jar` hat, wenn die neue Gruppe von Schnittstellen im Paket `com.ibm.mq.exits` verwendet wird, gibt es 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 Schnittstelle MQSendExit implementiert (für einen in Java geschriebenen Sendeexit).
- Eine Instanz der MQExternalSend-Exitklasse (für einen Sendeexit, der nicht in Java geschrieben ist).
- 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 WebSphere 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 WebSphere MQ classes for Java Verwenden Sie die in IBM WebSphere MQintegrierte Komprimierungsfunktion.

Mithilfe der in IBM WebSphere 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 WebSphere MQ, in den Abschnitten Datenkomprimierung (COMPMSG) und Headerkomprimierung (COMPHDR).

Eine Anwendung der IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java verwendet wird, wird bei einer neuen Verbindungsanforderung nicht immer automatisch eine bereits hergestellte Verbindung gemeinsam genutzt.

Verbindungspooling in WebSphere MQ -Klassen für Java

WebSphere MQ Classes for Java ermöglicht die Wiederverwendung von Ersatzverbindungen in Pools.

WebSphere MQ Classes for Java bietet zusätzliche Unterstützung für Anwendungen, die mit mehreren Verbindungen zu WebSphere MQ -Warteschlangenmanagern arbeiten. 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.

WebSphere 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. Wenn der Pool aktiv ist, wenn WebSphere MQ Classes for Java ein MQQueueManager -Objekt erstellt, wird dieser Standardpool durchsucht und jede geeignete Verbindung wiederverwendet. 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 WebSphere 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.

WebSphere MQ Classes for Java stellt einen Standardverbindungspool bereit, den Sie dazu verwenden können. Um den Pool zu aktivieren, verwenden Sie eine der MQEnvironment.addConnectionPoolToken()-Methoden. Wenn Sie den Pool inaktivieren möchten, verwenden Sie MQEnvironment.removeConnectionPoolToken().

Die folgende Beispielanwendung 'MQApp2' weist dieselben Funktionen wie MQApp1 auf, verbindet sich jedoch nur einmal mit jedem Warteschlangenmanager.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);
```

```
        }
}
```

Die erste fett gedruckte Zeile aktiviert den Standardverbindungspool durch die Registrierung eines MQPoolToken-Objekts bei MQEnvironment.

Der MQQueueManager-Konstruktor durchsucht jetzt diesen Pool nach einer geeigneten Verbindung. Er erstellt nur dann eine Verbindung mit dem Warteschlangenmanager, wenn er keine bereits vorhandene Verbindung finden kann. Der Aufruf qmgr.disconnect() gibt die Verbindung zur späteren Wiederverwendung an den Pool zurück. Diese API-Aufrufe sind die gleichen wie bei der Beispielanwendung MQApp1.

Die zweite hervorgehobene Zeile inaktiviert den Standardverbindungspool. Dadurch werden alle im Pool gespeicherten Warteschlangenmanagerverbindungen gelöscht. Dies ist wichtig, da andernfalls die Anwendung mit mehreren Live-Warteschlangenmanagerverbindungen im Pool beendet würde. Diese Situation könnte Fehler verursachen, die in den Warteschlangenmanagerprotokollen angezeigt würden.

Wenn eine Anwendung eine Definitionstabelle für den Clientkanal (CCDT) verwendet, um eine Verbindung zu einem Warteschlangenmanager herzustellen, durchsucht der MQQueueManager-Konstruktor zuerst die Tabelle nach einer geeigneten Definition eines Clientverbindungskanals. Falls eine gefunden wird, durchsucht der Konstruktor den Standardverbindungspool nach einer Verbindung, die für den Kanal verwendet werden kann. Wenn der Konstruktor keine geeignete Verbindung im Pool finden kann, durchsucht er danach die Definitionstabelle für den Clientkanal nach der nächsten geeigneten Definition eines Clientverbindungskanals und fährt wie zuvor beschrieben fort. Wenn der Konstruktor seine Suche in der Definitionstabelle für den Clientkanal abgeschlossen hat und keine geeignete Verbindung im Pool findet, beginnt er in der Tabelle einen zweiten Suchlauf. Während dieser Suche versucht der Konstruktor der Reihe nach, eine neue Verbindung für jede geeignete Definition eines Clientverbindungskanals zu erstellen. Er verwendet die erste Verbindung, die er erstellen kann.

Der Standardverbindungspool speichert bis zu zehn nicht verwendete Verbindungen und hält diese bis zu fünf Minuten lang aktiv. Die Anwendung kann dies ändern (Details hierzu finden Sie unter „Anderen Verbindungspool in WebSphere MQ Classes for Java bereitstellen” auf Seite 738).

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 WebSphere MQ -Klassen für Java

Dieses Beispiel zeigt, wie MQPoolTokens aus einer statischen Gruppe von registrierten MQPoolToken-Objekten hinzugefügt oder entfernt werden.

MQEnvironment verfügt über eine statische Gruppe von registrierten MQPoolToken-Objekten. Sie können MQPoolTokens mit den folgenden Methoden aus dieser Gruppe hinzufügen bzw. entfernen:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Eine Anwendung kann aus vielen Komponenten bestehen, die unabhängig voneinander vorhanden sind und mithilfe eines Warteschlangenmanagers Arbeiten durchführen. In einer solchen Anwendung sollte jede Komponente für ihre Lebensdauer ein MQPoolToken zur MQEnvironment-Gruppe hinzufügen.

Die Beispielanwendung MQApp3 erstellt beispielsweise zehn Threads und startet jeden einzelnen Thread. Jeder Thread registriert sein eigenes MQPoolToken, wartet eine gewisse Zeit und verbindet sich dann mit dem Warteschlangenmanager. Nach der Trennung des Threads entfernt er sein eigenes MQPoolToken.

Der Standardverbindungspool bleibt aktiv, solange sich mindestens ein Token in der Gruppe der MQPool-Tokens befindet, er bleibt also für die Dauer dieser Anwendung aktiv. Die Anwendung muss kein Masterobjekt in der Gesamtsteuerung der Threads aufbewahren.

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

Anderen Verbindungspool in WebSphere MQ Classes for Java bereitstellen

Dieses Beispiel zeigt, wie die Klasse **com.ibm.mq.MQSimpleConnectionManager** für die Bereitstellung eines anderen Verbindungspools verwendet wird.

Diese Klasse stellt grundlegendes Funktionen für das Verbindungspooling bereit und Anwendungen können diese Klasse verwenden, um das Verhalten des Pools anzupassen.

Sobald er instanziiert wurde, kann ein MQSimpleConnectionManager im MQQueueManager-Konstruktor angegeben werden. Der MQSimpleConnectionManager verwaltet dann die Verbindung, die dem erstellten MQQueueManager zugrunde liegt. Wenn der MQSimpleConnectionManager eine geeignete gepoolte Verbindung enthält, wird diese Verbindung wiederverwendet und nach einem Aufruf des Typs MQQueueManager.disconnect() an den MQSimpleConnectionManager zurückgegeben.

Das folgende Codefragment veranschaulicht dieses Verhalten:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

Die Verbindung, die während des ersten MQQueueManager-Konstruktors aufgebaut wird, wird nach dem Aufruf qmgr.disconnect() in myConnMan gespeichert. Die Verbindung wird dann beim zweiten Aufruf an den MQQueueManager-Konstruktor wiederverwendet.

Die zweite Zeile aktiviert den MQSimpleConnectionManager. Die letzte Zeile inaktiviert den MQSimpleConnectionManager und löscht damit sämtliche Verbindungen, die sich im Pool befinden. Ein MQSimpleConnectionManager befindet sich standardmäßig im Modus MODE_AUTO, der an späterer Stelle in diesem Abschnitt beschrieben wird.

Ein MQSimpleConnectionManager ordnet die Verbindungen, die am häufigsten verwendet werden, zu und löscht die Verbindungen, die am seltensten verwendet werden. Eine Verbindung wird standardmäßig gelöscht, wenn sie fünf Minuten lang nicht verwendet wurde oder wenn der Pool mehr als zehn nicht verwendete Verbindungen enthält. Sie können diese Werte durch das Aufrufen von MQSimpleConnectionManager.setTimeout() ändern.

Sie können einen MQSimpleConnectionManager auch als Standardverbindungspool konfigurieren, der verwendet wird, wenn im MQQueueManager-Konstruktor kein Connection Manager bereitgestellt wird.

Dies wird durch die folgende Anwendung veranschaulicht:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Mit den fett gedruckten Zeilen wird ein MQSimpleConnectionManager-Objekt erstellt und konfiguriert. Die Konfiguration bewirkt die folgenden Aktionen:

- Sie beendet Verbindungen, die eine Stunde lang nicht verwendet wurden.
- Sie begrenzt die Anzahl der von myConnMan verwalteten Verbindungen auf 75.
- Sie begrenzt die Anzahl der nicht verwendeten Verbindungen im Pool auf 50.
- Sie legt die Standardeinstellung MODE_AUTO fest. Dies bedeutet, dass der Pool nur aktiv ist, wenn es sich um den Standardverbindungsmanager handelt und die MQPoolTokens-Gruppe von MQEnvironment mindestens ein Token enthält.

Der neue MQSimpleConnectionManager wird dann als Standardverbindungsmanager festgelegt.

In der letzten Zeile ruft die Anwendung MQApp3.main() auf. Dadurch wird eine Reihe von Threads ausgeführt, wobei jeder Thread WebSphere MQ unabhängig voneinander verwendet. Diese Threads verwenden beim Aufbau von Verbindungen myConnMan.

Eigenen ConnectionManager für WebSphere MQ Classes for Java bereitstellen

WebSphere MQ Classes for Java stellt eine Teilimplementierung der Java EE Connector Architecture bereit, sodass Implementierungen von javax.resource.spi.ConnectionManager verwendet werden können.

Anwendungen und Anbieter von Middleware können alternative Implementierungen von Verbindungspools bereitstellen. WebSphere MQ Classes for Java stellt eine Teilimplementierung der Java EE Connector Architecture bereit. Implementierungen von **javax.resource.spi.ConnectionManager** können entweder als Standardverbindungsmanager verwendet oder im MQQueueManager-Konstruktor angegeben werden.

WebSphere MQ Classes for Java entspricht dem Verbindungsmanagementvertrag der Java EE Connector Architecture. Lesen Sie diesen Abschnitt zusammen mit dem Connection Management Contract der Java EE Connector Architecture (siehe Java-Website von Sun unter https://java.sun.com).

Die ConnectionManager-Schnittstelle definiert nur eine Methode:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

Der MQQueueManager-Konstruktor ruft allocateConnection im entsprechenden ConnectionManager auf. Er übergibt geeignete Implementierungen von ManagedConnectionFactory und ConnectionRequestInfo als Parameter zum Beschreiben der erforderlichen Verbindung.

Der ConnectionManager durchsucht seinen Pool nach einem javax.resource.spi.ManagedConnection-Objekt, das mit identischen ManagedConnectionFactory- und ConnectionRequestInfo-Objekten erstellt wurde. Wenn der ConnectionManager geeignete ManagedConnection-Objekte findet, erstellt er ein java.util.Set, das die möglichen ManagedConnections enthält. Anschließend ruft der ConnectionManager Folgendes auf:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

Die WebSphere MQ-Implementierung von ManagedConnectionFactory ignoriert den Parameter 'subject'. Diese Methode wählt aus der Gruppe eine geeignete ManagedConnection aus, die sie zurückgibt, oder sie gibt den Wert null zurück, wenn keine geeignete ManagedConnection gefunden wird. Wenn keine geeignete ManagedConnection im Pool vorhanden ist, kann der ConnectionManager auf die folgende Weise eine erstellen:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

Erneut wird der Parameter 'subject' ignoriert. Diese Methode stellt eine Verbindung zu einem WebSphere MQ-Warteschlangenmanager her und gibt eine Implementierung von 'javax.resource.spi.ManagedConnection' zurück, die die neu aufgebaute Verbindung darstellt. Sobald der ConnectionManager eine ManagedConnection erhalten hat (entweder aus dem Pool oder neu erstellt), erstellt er auf die folgende Weise eine Verbindungskennung:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

Diese Verbindungskennung kann von allocateConnection() zurückgegeben werden.

Ein ConnectionManager muss auf die folgende Weise Interesse an der ManagedConnection anmelden:

```
mc.addConnectionEventListener()
```

Der ConnectionEventListener wird benachrichtigt, wenn bei der Verbindung ein schwerwiegender Fehler auftritt oder wenn MQQueueManager.disconnect() aufgerufen wird. Wenn MQQueueManager.disconnect() aufgerufen wird, kann der ConnectionEventListener eine der folgenden Aktionen ausführen:

- Die ManagedConnection mithilfe des mc.cleanup()-Aufrufs zurücksetzen und sie dann an den Pool zurückgeben
- Die ManagedConnection mithilfe des mc.destroy()-Aufrufs löschen

Wenn der ConnectionManager der Standard-ConnectionManager ist, kann er auch Interesse am Status der von MQEnvironment verwalteten Gruppe von MQPoolTokens anmelden. Erstellen Sie dazu zunächst ein MQPoolServices-Objekt und registrieren Sie dann ein MQPoolServicesEventListener-Objekt im MQPoolServices-Objekt:

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

Der Listener wird benachrichtigt, wenn ein MQPoolToken zur Gruppe hinzugefügt oder daraus entfernt wird oder wenn der Standard-ConnectionManager geändert wird. Das MQPoolServices-Objekt bietet zudem eine Möglichkeit, die aktuelle Größe der Gruppe der MQPoolTokens abzufragen.

JTA/JDBC -Koordination mit WebSphere MQ -Klassen für Java

WebSphere MQ Classes for Java unterstützt die Methode MQQueueManager.begin (), die es WebSphere MQ ermöglicht, als Koordinator für eine Datenbank zu fungieren, die einen mit JDBC Typ 2 oder JDBC Typ 4 kompatiblen Treiber bereitstellt.

Diese Unterstützung ist nicht auf allen Plattformen verfügbar. Im Abschnitt https://www.ibm.com/software/integration/wmq/requirements/ 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.

V 7.5.0.7 Die bereitgestellte Datei `jdbcora12.dll` ist mit Oracle 12C kompatibel und für eine Serverinstallation mit IBM WebSphere 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 WebSphere MQ-Klassen für Java. Sie müssen die XA-Bibliothek, die auch als Switchloaddatei beschrieben wird, beim Warteschlangenmanager deklarieren. Verwenden Sie IBM WebSphere 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.

WebSphere 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 Makefile verknüpfen, die mit WebSphere 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 WebSphere 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 WebSphere 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. Wenn die Konfigurationsdatei (qm.ini) des Warteschlangenmanagers bereits XAResourceManager-Zeilengruppen für DB2- oder Oracle-Datenbanken enthält, müsse Sie den Eintrag 'SwitchFile' in den einzelnen Zeilengruppen wie folgt 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 Verwaltung . 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

Es bestehen bestimmte Probleme und Einschränkungen hinsichtlich der JTA/JDBC-Unterstützung. Einige davon hängen vom verwendeten Datenbankmanagementsystem ab.

Da diese 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. Vermeiden Sie **immer** eine abrupte Beendigung der Datenbank, solange Anwendungen über offene Verbindungen mit ihr verfügen.

Mehrere XAResourceManager-Zeilengruppen

Die Verwendung mehrerer XAResourceManager-Zeilengruppen in einer Konfigurationsdatei des Warteschlangenmanagers (qm.ini) wird nicht unterstützt. Mit Ausnahme der ersten XAResourceManager-Zeilengruppe werden alle übrigen Zeilengruppen ignoriert.

DB2

Gelegentlich gibt DB2 einen Fehler vom Typ SQL0805N 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 Dokumentation zu DB2.

Die XAResourceManager-Zeilengruppe muss für die Verwendung von ThreadOfControl=PROCESS konfiguriert werden. Da dies bei DB2 Version 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
```

Dies verhindert nicht, dass die Java-Anwendungen, die die JTA/JDBC -Koordination verwenden, selbst als Multithread-Prozess 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().

Unterstützung für Secure Sockets Layer (SSL) in WebSphere MQ Classes for Java

WebSphere MQ Classes for Java-Clientanwendungen unterstützen die SSL-Verschlüsselung (Secure Sockets Layer). Für die Verwendung der SSL-Verschlüsselung benötigen Sie einen JSSE-Provider.

WebSphere MQ -Klassen für Java-Clientanwendungen, die TRANSPORT (CLIENT) verwenden, unterstützen die SSL-Verschlüsselung (Secure Sockets Layer). SSL stellt eine Kommunikationsverschlüsselung, Authentifizierung und Nachrichtenintegrität bereit. In der Regel wird damit die Kommunikation zwischen zwei Peers im Internet oder innerhalb eines Intranets geschützt.

WebSphere MQ Classes for Java verwendet Java Secure Socket Extension (JSSE) für die SSL-Verschlüsselung und erfordert 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 WebSphere MQ Classes for Java-Clientanwendung eine Definitionstabelle für Clientkanäle (CCDT) verwendet, um eine Verbindung zu einem Warteschlangenmanager herzustellen, lesen Sie den Abschnitt „Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for Java verwenden” auf Seite 714.

SSL in IBM WebSphere MQ classes for Java aktivieren

Für die Aktivierung von SSL müssen Sie eine Cipher-Suite angeben. Für die Angabe einer Cipher-Suite haben Sie zwei Möglichkeiten.

SSL wird nur für Clientverbindungen unterstützt. Zur Aktivierung von SSL müssen Sie die Cipher-Suite angeben, die bei der Kommunikation mit dem Warteschlangenmanager verwendet werden soll. Diese Cipher-Suite muss der im Zielkanal festgelegten CipherSpec entsprechen. Außerdem muss die angegebene Cipher-Suite von Ihrem JSEE-Provider unterstützt werden. Da sich Cipher-Suites jedoch von CipherSpecs unterscheiden, haben sie unterschiedliche Namen. „SSL CipherSpecs und CipherSuites in WebSphere MQ Classes for Java” auf Seite 749 enthält eine Tabellenzuordnung zwischen den von IBM WebSphere MQ unterstützten CipherSpecs und ihren Cipher-Suite-Entsprechungen, die in JSSE bekannt sind.

Geben Sie für die SSL-Aktivierung mithilfe der statischen Member-Variablen sslCipherSuite von MQEnvironment die Cipher-Suite an. Das folgende Beispiel bezieht sich auf einen SVRCONN-Kanal mit dem Namen SECURE.SVRCONN.CHANNEL, der konfiguriert wurde, um SSL mit der CipherSpec 'RC4_MD5_EXPORT' anzufordern:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Obwohl der Kanal über die CipherSpec 'RC4_MD5_EXPORT' verfügt, muss die Java-Anwendung die Cipher-Suite 'SSL_RSA_EXPORT_WITH_RC4_40_MD5' angeben. Im Abschnitt „SSL CipherSpecs und CipherSuites in WebSphere MQ Classes for Java” auf Seite 749 finden Sie eine Liste der Zuordnungen zwischen den CipherSpecs und den Cipher-Suites.

Eine Anwendung kann eine Cipher-Suite auch durch die Festlegung der Umgebungseigenschaft CMQC.SSL_CIPHER_SUITE_PROPERTY angeben.

Alternativ dazu können Sie die Definitionstabelle für den Clientkanal (Client Channel Definition Table, CCDT) verwenden. Weitere Informationen finden Sie unter „Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for Java verwenden" auf Seite 714

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 WebSphere 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.

Damit unter Verwendung von SSL eine Verbindung erfolgreich hergestellt werden kann, muss der JSSE-Truststore mit Stammzertifikaten der Zertifizierungsstelle konfiguriert werden, damit das vom Warteschlangenmanager vorgelegte Zertifikat authentifiziert werden kann. Ähnlich gilt Folgendes: Wenn SSLClientAuth im SVRCONN-Kanal auf MQSSL_CLIENT_AUTH_REQUIRED gesetzt wurde, muss der JSSE-Schlüsselspeicher ein Identifizierungszertifikat enthalten, das vom Warteschlangenmanager als vertrauenswürdig eingestuft wird.

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für UNIX, Linux und Windows

Definierten Namen des Warteschlangenmanagers in IBM WebSphere MQ classes for Java verwenden

Der Warteschlangenmanager weist sich mithilfe eines SSL-Zertifikats aus, das einen definierten Namen (DN) enthält. Eine IBM WebSphere MQ classes for Java-Clientanwendung kann diesen definierten Namen verwenden, um sicherzustellen, dass sie mit dem richtigen Warteschlangenmanager kommuniziert.

Ein definiertes Namensmuster wird mit der Variablen 'sslPeerName' von MQEnvironment angegeben. Sie können beispielsweise Folgendes festlegen:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Die Verbindung kann nur erfolgreich hergestellt werden, wenn der Warteschlangenmanager ein Zertifikat mit einem allgemeinen Namen vorlegt, der mit 'QMGR.' beginnt. und mindestens zwei Namen von Organisationseinheiten, von denen der erste IBM und der zweite WebSpheresein muss

Wenn sslPeerName festgelegt wird, können Verbindungen nur dann erfolgreich hergestellt werden, wenn ein gültiges Muster festgelegt wird und der Warteschlangenmanager ein entsprechendes Zertifikat vorlegt.

Eine Anwendung kann den definierten Namen des Warteschlangenmanagers auch durch die Festlegung der Umgebungseigenschaft CMQC.SSL_PEER_NAME_PROPERTY angeben. Weitere Informationen zu definierten Namen finden Sie im Abschnitt Definierte Namen.

Zertifikatswiderrufslisten in IBM WebSphere MQ classes for Java verwenden

Geben Sie die durch die java.security.cert.CertStore-Klasse zu verwendenden Zertifikatswiderrufslisten an.IBM WebSphere MQ classes for Java Anschließend werden die Zertifikate anhand der angegebenen CRL überprüft.

Eine Zertifikatswiderrufsliste (Certificate Revocation List, CRL) ist eine Gruppe von Zertifikaten, die durch die ausstellende Zertifizierungsstelle oder durch die lokale Organisation widerrufen wurden. CRLs wer-

den 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 WebSphere MQfinden Sie unter Arbeiten mit Zertifikatswiderrufslisten und Berechtigungswiderrufslisten und Zugreifen auf CRLs und ARLs mit WebSphere MQ Classes for Java und WebSphere MQ Classes for JMS.

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 WebSphere MQ -Klassen für Java neu vereinbaren

Eine WebSphere MQ -Clientanwendung für Java kann steuern, wann der geheime Schlüssel, der für die Verschlüsselung einer Clientverbindung verwendet wird, neu vereinbart wird, gemessen an der Gesamtzahl der gesendeten und empfangenen Byte.

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 sslResetoder der Umgebungseigenschaft MQC.SSL_RESET_COUNT_PROPERTY stellt die Gesamtzahl der Bytes dar, die vom WebSphere MQ Classes for Java-Clientcode gesendet und empfangen wurden, bevor der geheime Schlüssel neu 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 Bytes umfasst auch Steuerinformationen, die vom WebSphere MQ -Client gesendet und empfangen werden.

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 WebSphere 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 WebSphere MQ classes for Java verwenden diese SSLSocketFactory, um eine Verbindung zum IBM WebSphere MQ-Warteschlangenmanager herzustellen. 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-Keystore oder -Truststore in WebSphere 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 den Inhalt des JSSE-Keystores oder -Truststores ändern oder die Position der Keystore-oder Truststore-Datei ändern, übernehmen WebSphere MQ -Klassen für Java-Anwendungen, die zu diesem Zeitpunkt ausgeführt werden, die Änderungen nicht automatisch. 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 SSL mit WebSphere MQ -Klassen für Java

Eine Reihe von Ursachencodes kann von WebSphere MQ -Klassen für Java ausgegeben werden, wenn eine Verbindung zu einem Warteschlangenmanager über SSL hergestellt wird.

Diese werden in der folgenden Liste erläutert:

MQRC_SSL_NOT_ALLOWED
Die sslCipherSuite-Eigenschaft wurde festgelegt, es wurde jedoch eine Bindungsverbindung verwendet. Nur Clientverbindungen unterstützen SSL.

MQRC_JSSE_ERROR
Der JSSE-Provider hat einen Fehler gemeldet, der nicht von WebSphere 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 SSL-Konfigurationsoptionen wurde ausgegeben, bei der Initialisierung der SSL-Umgebung ist aber 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 WebSphere MQ verwendet werden können, im Abschnitt „SSL CipherSpecs und CipherSuites in WebSphere MQ Classes for Java” auf Seite 749.

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.

SSL CipherSpecs und CipherSuites in WebSphere MQ Classes for Java

Ob Anwendungen, die die IBM WebSphere MQ classes for Java verwenden, eine Verbindung zu einem Warteschlangenmanager herstellen können, hängt von der am Serverende des MQI-Kanals angegebenen CipherSpec und von der am Clientende angegebenen Cipher-Suite ab.

Ob eine Anwendung, die die IBM WebSphere MQ classes for Java verwendet, eine Verbindung zu einem Warteschlangenmanager herstellen kann, hängt bei jeder Kombination von CipherSpec und Cipher-Suite vom Wert des Feldes 'sslFipsRequired' in der MQEnvironment-Klasse oder vom Wert der Umgebungseigenschaft CMQC.SSL_FIPS_REQUIRED_PROPERTY ab.

Am Serverende eines MQI-Kanals kann der Name einer CipherSpec als Wert des Parameters "SSLCIPH" in einem Befehl "DEFINE CHANNEL CHLTYPE(SVRCONN)" angegeben werden. Am Clientende eines MQI-Kanals kann eine Anwendung von IBM WebSphere MQ classes for Java das Feld sslCipherSuite in der MQEnvironment-Klasse oder die Umgebungseigenschaft CMQC.SSL_CIPHER_SUITE_PROPERTYfestlegen.

Anwendung für die Verwendung von IBM Java- oder Oracle Java-Cipher-Suite-Zuordnungen konfigurieren

Ab IBM WebSphere MQ Version 7.5.0Fixpack 5können Sie konfigurieren, ob Ihre Anwendung die Standardzuordnungen von IBM Java CipherSuite zu WebSphere MQ CipherSpec oder die Zuordnungen von Oracle CipherSuite zu WebSphere MQ CipherSpec verwendet. Daher können Sie TLS-Cipher-Suites unabhängig davon verwenden, 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 IBM Java-Cipher-Suite/WebSphere MQ-CipherSpec-Zuordnungen werden verwendet.

Dies ist der Standardwert.

false
Die Oracle-Cipher-Suite/WebSphere MQ-CipherSpec-Zuordnungen werden verwendet.

In der folgenden Tabelle sind die von IBM WebSphere MQ unterstützten CipherSpecs sowie die entsprechenden Cipher-Suites aufgeführt. In der Tabelle ist auch angegeben, ob eine Anwendung, die die IBM WebSphere MQ classes for Java verwendet, eine Verbindung zu einem Warteschlangenmanager herstellen kann, wenn am Serverende des MQI-Kanals eine CipherSpec angegeben ist und am Clientende die entsprechende Cipher-Suite angegeben ist.

Tabelle 89. Von WebSphere MQ unterstützte CipherSpecs und die entsprechenden Cipher-Suites

CipherSpec	**Entsprechende Cipher-Suite**	**Verbindung möglich, wenn SFIPS [1] auf YES gesetzt ist?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	Nein
NULL_SHA	SSL_RSA_WITH_NULL_SHA	Nein
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5 (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	Nein
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (IBM JRE) SSL_RSA_EXPORT_WITH_RC4_40_MD5 (Oracle JRE)	Nein
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256 (IBM JRE) TLS_RSA_WITH_NULL_SHA256 (Oracle JRE)	Nein [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA (Oracle JRE)	Ja [5] [7]
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA256 (Oracle JRE)	Ja [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA (Oracle JRE)	Ja [5] [7]

Tabelle 89. Von WebSphere MQ unterstützte CipherSpecs und die entsprechenden Cipher-Suites (Forts.)		
CipherSpec	**Entsprechende Cipher-Suite**	**Verbindung möglich, wenn SFIPS [1] auf YES gesetzt ist?**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA256 (Oracle JRE)	Ja[5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8]	SSL_RSA_WITH_DES_CBC_SHA	Nein[3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[9] [9]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Ja
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein[4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (IBM JRE) Keine Entsprechung für Oracle JRE.	Nein[6]

Anmerkungen:

1. Geben Sie in einer Anwendung, die die IBM WebSphere MQ classes for Java verwendet, an, dass nur FIPS-zertifizierte Algorithmen verwendet werden sollen, indem Sie für das Feld 'sslFipsRequired' in der MQEnvironment-Klasse `true` festlegen. Geben Sie an, dass auch nicht FIPS-zertifizierte Algorithmen verwendet werden können, indem Sie für das Feld 'sslFipsRequired' `false` festlegen. Alternativ können Sie die Umgebungseigenschaft CMQC.SSL_FIPS_REQUIRED_PROPERTY festlegen.
2. Für diese CipherSpec ist keine funktional entsprechende Cipher-Suite vorhanden.
3. Diese CipherSpec wurde vor dem 19. Mai 2007 nach FIPS 140-2 zertifiziert.
4. Diese CipherSpec wurde vor dem 19. Mai 2007 nach FIPS 140-2 zertifiziert. Der Name FIPS_WITH_DES_CBC_SHA ist historisch und gibt die Tatsache wieder, dass diese Verschlüsselungsspezifikation (CipherSpec) zuvor mit FIPS konform war. Dies ist jedoch nicht mehr der Fall. Diese CipherSpec ist veraltet und sollte nicht mehr verwendet werden.
5. Diese CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) können nicht zum Sichern einer Verbindung von WebSphere MQ Explorer zu einem Warteschlangenmanager verwendet werden, es sei denn, die entsprechenden uneingeschränkten Richtliniendateien werden auf die vom Explorer verwendete JRE angewendet.

   Weitere Informationen zu Richtliniendateien finden Sie in den Sicherheitsinformationen.
6. Der Name FIPS_WITH_3DES_EDE_CBC_SHA ist historisch und gibt die Tatsache wieder, dass diese Verschlüsselungsspezifikation (CipherSpec) zuvor mit FIPS konform war. Dies ist jedoch nicht mehr der Fall. Diese CipherSpec ist veraltet und sollte nicht mehr verwendet werden.
7. Für diese CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) ist IBM JREs 6.0 SR13 FP2 , 7.0 SR4 FP2 erforderlich.
8. Für diese CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) kann SSLv3 oder TLS verwendet werden. Wenn FIPS nicht aktiviert ist, wird standardmäßig SSLv3 verwendet. Wenn TLS verwendet werden soll, muss die Java-Systemeigenschaft **`com.ibm.mq.cfg.preferTLS`** auf `true` gesetzt werden.

9. Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Zugehörige 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

MQdev Blog: MQ Java, TLS Ciphers, Non-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837

MQdev Blog: The relationship between MQ CipherSpecs and Java Cipher Suites

WebSphere MQ -Klassen für Java-Anwendungen ausführen

Wenn Sie eine Anwendung (eine Klasse, die eine Methode main () enthält) im Client-oder im Bindungsmodus schreiben, führen Sie Ihr Programm mit dem Java-Interpreter aus.

Verwenden Sie den folgenden Befehl:

```
java -Djava.library.path=library_path MyClass
```

Dabei ist *bibliothekspfad* der Pfad zu den WebSphere MQ -Klassen für Java-Bibliotheken (siehe WebSphere MQ -Klassen für Java-Bibliotheken).

WebSphere MQ Classes for Java-umgebungsabhängiges Verhalten

Mit WebSphere MQ Classes for Java können Sie Anwendungen erstellen, die für verschiedene Versionen von WebSphere MQausgeführt werden können. In dieser Themensammlung wird das Verhalten der Java-Klassen in Abhängigkeit von diesen verschiedenen Versionen beschrieben.

WebSphere MQ Classes for Java stellt einen Kern von Klassen bereit, die in allen Umgebungen konsistente Funktionen und konsistentes Verhalten bereitstellen. 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 den Referenzinformationen zur Anwendungsprogrammierung.

Kernklassen in WebSphere MQ Classes for Java

WebSphere MQ Classes for Java enthält eine Kerngruppe von Klassen, die in allen Umgebungen verwendet werden kann.

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 für Kernklassen von WebSphere MQ -Klassen für Java” auf Seite 753 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 für Kernklassen von WebSphere MQ -Klassen für Java” auf Seite 753); 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. (Siehe „Verbindungsoptionen für WebSphere MQ -Klassen für Java” auf Seite 694.)

Einschränkungen und Varianten für Kernklassen von WebSphere MQ -Klassen für 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 entspricht dem bei Verwendung eines Windows-, UNIX -oder Linux WebSphere MQ -Warteschlangenmanagers, mit Ausnahme der folgenden geringfügigen Einschränkungen und Varianten.

*Einschränkungen für MQGMO_ * -Werte in WebSphere MQ -Klassen für 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 es von Java verwendet wird.

*Einschränkungen für MQPMRF_ * -Werte in WebSphere MQ -Klassen für 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 für MQPMO_ * -Werte in WebSphere MQ -Klassen für 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 Variationen für MQCNO_ * -Werte in WebSphere MQ -Klassen für Java*
Bestimmte MQCNO_*-Werte werden nicht unterstützt.

- Die automatische Wiederherstellung einer Clientverbindung wird von WebSphere MQ-Klassen für 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 für MQRO_ * -Werte in WebSphere MQ -Klassen für 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 finden Sie unter Report.

Funktionen außerhalb der Kernklassen von WebSphere MQ Classes for Java

WebSphere MQ -Klassen für Java enthalten bestimmte Funktionen, die speziell für die Verwendung von API-Erweiterungen entwickelt wurden, die nicht von allen Warteschlangenmanagern unterstützt werden. In dieser Themensammlung 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 WebSphere MQ-Warteschlangenmanager auf Systemen unter UNIX, Linux oder Windows im Bindungsmodus verwendet werden. Andernfalls schlägt sie mit MQRC_ENVIRONMENT_ERROR fehl.

Weitere Informationen finden Sie in „JTA/JDBC -Koordination mit WebSphere MQ -Klassen für Java” auf Seite 741.

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 WebSphere MQ Classes for Java

Nicht alle Warteschlangenmanager ermöglichen Ihnen das Öffnen einer MQDistributionList.

Die folgenden Klassen werden für die Erstellung von Verteilerlisten verwendet:

MQDistributionList
MQDistributionListItem
MQMessageTracker

Sie können MQDistributionLists und MQDistributionListItems in jeder Umgebung erstellen und füllen, allerdings ermöglichen Ihnen nicht alle Warteschlangenmanager das Öffnen einer MQDistributionList. Insbesondere z/OS-Warteschlangenmanager unterstützen keine Verteilerlisten. Der Versuch, eine MQDistributionList bei Verwendung eines solchen Warteschlangenmanagers zu öffnen, führt zum Fehler MQRC_OD_ERROR..

Varianten in MQPutMessageOptions-Feldern
Wenn ein Warteschlangenmanager keine Verteilerlisten unterstützt, werden bestimmte MQPMO-Felder anders behandelt.

Vier Felder in MQPMO werden als die folgenden Elementvariablen in der MQPutMessageOptions-Klasse wiedergegeben:

knownDestCount
unknownDestCount
invalidDestCount
recordFields

Diese Felder sind in erster Linie für die Verwendung in Verbindung mit Verteilerlisten vorgesehen. Allerdings füllt ein Warteschlangenmanager, der Verteilerlisten unterstützt, auch die DestCount-Felder nach einem MQPUT-Vorgang in einer einzelnen Warteschlange aus. Wenn die Warteschlange beispielsweise in eine lokale Warteschlange aufgelöst wird, wird knownDestCount auf 1 gesetzt, während die übrigen beiden Zählerfelder auf 0 gesetzt werden.

Wenn der Warteschlangenmanager keine Verteilerlisten unterstützt, werden diese Werte wie folgt simuliert:

- Wenn die Methode put() erfolgreich ausgeführt wird, wird unknownDestCount auf 1 gesetzt, während die übrigen Felder auf 0 gesetzt werden.
- Wenn die Methode put() fehlschlägt, wird invalidDestCount auf 1 gesetzt, während die übrigen Felder auf 0 gesetzt werden.

Die Variable 'recordFields' wird in Verbindung mit Verteilerlisten verwendet. Ein Wert kann unabhängig von der Umgebung jederzeit in recordFields geschrieben werden. Er wird ignoriert, wenn das MQPutMessageOptions-Objekt nicht bei einem MQDistributionList.put()-Vorgang, sondern bei einem MQDestination.put()- oder MQQueueManager.put()-Vorgang verwendet wird.

Einschränkungen in MQMD-Feldern mit WebSphere 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 Nachrichtengruppierung und Segmentierung unterstützt.

Wenn eine Java-Anwendung versucht, () eine Nachricht von einem Warteschlangenmanager abzurufen, der diese Felder nicht unterstützt, und die abzurufende physische Nachricht Teil einer Gruppe segmentierter Nachrichten ist (d. h., sie enthält vom Standard abweichende Werte für die MQMD-Felder), wird sie ohne Fehler abgerufen. Allerdings werden die MQMD-Felder in der MQMessage nicht aktualisiert, die MQMessage-Formateigenschaft wird auf MQFMT_MD_EXTENSION gesetzt und die tatsächlichen Nachrichtendaten erhalten als Präfix eine MQMDE-Struktur, die die Werte für die neuen Felder enthält.

Einschränkungen für WebSphere MQ Classes for Java unter CICS Transaction Server

In einer Umgebung mit CICS Transaction Server for z/OS darf nur der Hauptthread (der erste Thread) CICS- oder WebSphere MQ-Aufrufe ausgeben.

Beachten Sie, dass die Verwendung von WebSphere 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.

Anwendungen, die IBM WebSphere MQ-Klassen für Java verwenden, in Java Platform, Enterprise Edition ausführen

Es gibt bestimmte Einschränkungen und Designaspekte, die berücksichtigt werden müssen, bevor IBM WebSphere MQ -Klassen für Java in Java EE verwendet werden.

IBM WebSphere MQ -Klassen für Java haben Einschränkungen, wenn sie in einer Java EE -Umgebung verwendet werden. Beim Entwerfen, Implementieren und Verwalten einer IBM WebSphere MQ -Klassen für Java -Anwendung, die in einer Java EE -Umgebung ausgeführt wird, müssen zusätzliche Aspekte berücksichtigt werden. Diese Einschränkungen und Aspekte werden in den folgenden Abschnitten kurz erläutert.

Einschränkungen bei JTA-Transaktionen

Für Anwendungen mit IBM WebSphere MQ-Klassen für Java wird als Transaktionsmanager nur IBM WebSphere MQ selbst unterstützt. Eine Anwendung unter JTA-Steuerung kann zwar IBM WebSphere MQ-Klassen für 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 für Anwendungs-oder Bean-gesteuerte Transaktionen sowie für containergesteuerte Transaktionen und alle Java EE -Container. Wenn eine Messaging-Bearbeitung direkt mit IBM WebSphere MQ innerhalb von serverkoordinierten Transaktionen der Anwendung durchgeführt werden soll, müssen stattdessen IBM WebSphere MQ-Klassen für JMS verwendet werden.

Threaderstellung

IBM WebSphere MQ-Klassen für 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 WebSphere MQ-Klassen für 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 (z. B. Anwendungen, die in EJB-und Webcontainern ausgeführt werden) dürfen keine neuen Threads erstellen. Stattdessen muss die gesamte Bearbeitung in den Hauptanwendungsthreads stattfinden, die vom Anwendungsserver verwaltet werden. Wenn Anwendungen IBM WebSphere MQ-Klassen für Java verwenden, ist der Anwendungsserver möglicherweise nicht in der Lage, zwischen dem Anwendungscode und dem Code der IBM WebSphere MQ-Klassen für Java zu unterscheiden. Daher führen die zuvor beschriebenen Threads dazu, dass die Anwendung nicht mit der Containerspezifikation konform ist. IBM WebSphere MQ Classes for JMS unterbricht diese Java EE -Spezifikationen nicht und kann daher stattdessen verwendet werden.

Sicherheitseinschränkungen

Die von einem Anwendungsserver implementierten Sicherheitsrichtlinien verhindern möglicherweise bestimmte Operationen, die von der API der IBM WebSphere MQ-Klassen für 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 WebSphere MQ-Klassen für 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 WebSphere MQ-Klassen für 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 Anwendungsisolation. Das Design und die Implementierung von IBM WebSphere MQ -Klassen für Java vor der Java EE -Umgebung. IBM WebSphere MQ Klassen für Java können auf eine Weise verwendet werden, die das Konzept der Anwendungsisolation nicht unterstützt. 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 SSL-Konfiguration für geschützte 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 Java EEmuss jede Anwendung ihre eigene Konfiguration verwenden, indem sie MQQueueManager -Objekte mit einer bestimmten Gruppe von Eigenschaften erstellt, anstatt standardmäßig die Eigenschaften zu verwenden, die in der prozessweiten MQEnvironment-Klasse konfiguriert sind.
- Die MQEnvironment-Klasse führt verschiedene statische Methoden ein, die global Aktionen für alle Anwendungen mit IBM WebSphere MQ-Klassen für Java innerhalb desselben JVM-Prozesses ausführen. Dieses Verhalten kann grundsätzlich nicht für bestimmte Anwendungen außer Kraft gesetzt werden. Beispiele:
  - die Konfiguration von SSL-Eigenschaften (beispielsweise die Position des Schlüsselspeichers)
  - 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

  Das Aufrufen 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 Verbindungspoolkonfiguration, wie z. B. das Ersetzen des Standardverbindungsmanagers für eine Anwendung mit der Methode MQEnvironment.setDefaultConnectionManager(), wirken sich daher auf andere Anwendungen aus, die in demselben Java EE -Anwendungsserver ausgeführt werden.
- SSL wird für Anwendungen mit IBM WebSphere MQ-Klassen für Java mithilfe der MQEnvironment-Klasse und über die MQQueueManager-Objekteigenschaften konfiguriert. Es wird nicht in die verwaltete Sicherheitskonfiguration des Anwendungsservers selbst integriert. Sie müssen sicherstellen, dass Sie die IBM WebSphere MQ-Klassen für Java entsprechend konfigurieren, damit Sie Ihre erforderliche Sicherheitsstufe erhalten. Verwenden Sie also nicht die Konfiguration des Anwendungsservers.

Einschränkungen beim Bindungsmodus

IBM WebSphere MQ und WebSphere Application Server können auf derselben Maschine installiert werden, sodass sich die Hauptversionen des Warteschlangenmanagers und des IBM WebSphere MQ -Ressourcenadapters (RA), die im Lieferumfang von WebSphere Application Server enthalten sind, unterscheiden. WebSphere Application Server Version 7.0, in dem ein IBM WebSphere MQ-RA der Version 7.0.1 enthalten ist, kann beispielsweise auf demselben System installiert sein wie ein Warteschlangenmanager der Version 6.0.

Wenn sich die Hauptversionen des Warteschlangenmanagers und Ressourcenadapters unterscheiden, können keine Bindungsverbindungen verwendet werden. Bei allen Verbindungen, die vom WebSphere Application Server zu dem Warteschlangenmanager hergestellt werden, der den Ressourcenadapter verwendet, müssen Verbindungen des Clienttyps verwendet werden. Wenn die Versionen identisch sind, können Bindungsverbindungen verwendet werden.

WebSphere MQ-Klassen für JMS verwenden

WebSphere MQ Classes for Java Message Service (WebSphere MQ Classes for JMS) ist der JMS-Anbieter, der mit WebSphere MQbereitgestellt wird. Neben der Implementierung der Schnittstellen, die im Paket 'javax.jms' definiert sind, stellen die WebSphere MQ-Klassen für JMS zwei Gruppen von Erweiterungen für die JMS-API zur Verfügung.

In der JMS-Spezifikation wird eine Gruppe von Schnittstellen definiert, die Anwendungen zur Durchführung von Messaging-Operationen verwenden können. Das Paket 'javax.jms' definiert die JMS-Schnittstellen und ein JMS-Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. WebSphere MQ Version 7.5 verwendet derzeit die Spezifikation JMS 1.1 . WebSphere MQ Classes for JMS ist ein JMS-Provider, der die JMS-Schnittstellen für WebSphere MQ implementiert.

Die JMS-Spezifikation setzt voraus, dass es sich bei den Objekten 'ConnectionFactory' und 'Destination' um verwaltete Objekte handelt. Ein Administrator erstellt und verwaltet verwaltete Objekte in einem zentralen Repository, und eine JMS-Anwendung ruft diese Objekte über JNDI (Java Naming and Directory Interface) ab. WebSphere MQ Classes for JMS unterstützt die Verwendung verwalteter Objekte und ein Administrator kann entweder das WebSphere MQ JMS-Verwaltungstool oder WebSphere MQ Explorer verwenden, um verwaltete Objekte zu erstellen und zu verwalten.

Darüber hinaus bieten WebSphere MQ-Klassen für 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.

Die WebSphere MQ JMS-Erweiterungen

Die Vorgängerreleases der WebSphere MQ-Klassen für JMS enthalten Erweiterungen, die in Objekten wie MQConnectionFactory-, MQQueue- und MQTopic-Objekten implementiert werden. Diese Objekte verfügen über Eigenschaften und Methoden, die für WebSphere MQ spezifisch sind. 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 WebSphere MQ-Klassen für JMS wurden diese Erweiterungen, die unter dem Begriff WebSphere MQ JMS-Erweiterungen bekannt sind, beibehalten. Sie können alle Anwendungen, die diese Erweiterungen verwenden, weiterhin ohne Änderung nutzen.

Die IBM JMS-Erweiterungen

Dieses Release der WebSphere MQ-Klassen für JMS stellt eine generisch gehaltene Gruppe von Erweiterungen für die JMS-API zur Verfügung, die sich nicht speziell auf WebSphere MQ als Messaging-System beziehen. Diese Erweiterungen werden als IBM JMS-Erweiterungen bezeichnet und haben die folgende allgemeine Zielsetzung:

- Bereitstellung höherer Konsistenz bei den verschiedenen IBM JMS-Providern
- Vereinfachung des Schreibens einer Brückenanwendung zwischen zwei IBM Messaging-Systemen
- Vereinfachung des Portierens einer Anwendung zwischen zwei IBM JMS-Providern

Die Erweiterungen bieten Funktionalität, die der von Message Service Client for C/C++ und Message Service Client for .NET vergleichbar ist.

Warum sollten Sie WebSphere MQ-Klassen für JMS verwenden?

Die Verwendung von WebSphere MQ-Klassen für JMS bietet die folgenden Vorteile:

- JMS-Kenntnisse können wiederverwendet werden.

  WebSphere MQ-Klassen für JMS ist ein JMS-Provider, der die JMS-Schnittstellen für WebSphere MQ als Messaging-System implementiert. Falls WebSphere MQ in Ihrem Unternehmen bislang noch nicht eingesetzt wurde, Sie aber Kenntnisse im Bereich der JMS-Anwendungsentwicklung besitzen, ist es für Sie unter Umständen einfacher, die vertraute JMS-API anstelle der anderen, mit WebSphere MQ bereitgestellten APIs für den Zugriff auf die WebSphere MQ-Ressourcen zu verwenden.

- JMS ist ein integraler Bestandteil von Java Platform, Enterprise Edition (Java EE).

  JMS ist die natürliche API für Messaging auf der Java EE -Plattform. Jeder Java EE -konforme Anwendungsserver muss einen JMS-Provider enthalten. Sie können JMS in Anwendungsclients, Servlets, JavaServer -Seiten (JSPs), Enterprise JavaBeans (EJBs) und Message-driven Beans (MDBs) verwenden. Beachten Sie insbesondere, dass Java EE -Anwendungen MDBs für die asynchrone Verarbeitung von Nachrichten verwenden und alle Nachrichten als JMS-Nachrichten an MDBs zugestellt werden.

- Ein Administrator kann verwaltete JMS-Objekte in einem zentralen Repository erstellen und verwalten und Anwendungen von WebSphere MQ Classes for JMS können diese Objekte über JNDI (Java Naming and Directory Interface) abrufen.

  JMS-Verbindungsfactorys und -Ziele binden WebSphere MQ-spezifische Informationen wie Namen von Warteschlangenmanagern, Kanalnamen, Verbindungsoptionen, Warteschlangennamen und Themennamen ein. Wenn Verbindungsfactorys und Ziele als verwaltete Objekte gespeichert werden, werden diese Informationen in einer Anwendung nicht fest codiert. Dank dieses Konzepts bleibt die Anwendung bis zu einem gewissen Grad unabhängig von der zugrunde liegenden WebSphere MQ-Konfiguration.

- JMS ist eine API gemäß Branchenstandard, die Portierbarkeit von Anwendungen ermöglicht.

  Eine JMS-Anwendung kann über die JNDI Verbindungsfactorys und Ziele abrufen, die als verwaltete Objekte gespeichert sind, und nur die im Paket 'javax.jms' definierten Schnittstellen zur Durchführung von Messaging-Operationen verwenden. Die Anwendung ist dann völlig unabhängig von einem bestimmten JMS-Provider wie WebSphere MQ-Klassen für JMS und kann von einem JMS-Provider an 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 WebSphere MQ-Klassen für JMS mithilfe der JMS-API-Erweiterungen Verbindungsfactorys und Ziele während der Laufzeit dynamisch erstellen und konfigurieren. Die Anwendung ist dann völlig eigenständig, jedoch an WebSphere MQ-Klassen für JMS als JMS-Provider gebunden.

- Brückenanwendungen sind unter Verwendung von JMS möglicherweise einfacher zu 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. Stattdessen können Sie eine Brückenanwendung schreiben, indem Sie zwei JMS-Provider verwenden, für jedes Messaging-System einen. Die Anwendung verwendet dann nur eine API, die JMS-API, und verarbeitet nur JMS-Nachrichten.

Einführung in die WebSphere MQ-Klassen für JMS

Dieser Abschnitt enthält eine Übersicht über die WebSphere MQ-Klassen für JMS und enthält alle wichtigen Punkte, die Sie vor der Verwendung der WebSphere MQ-Klassen für JMS wissen müssen.

Voraussetzungen für WebSphere MQ Classes for JMS

Für die Entwicklung und Ausführung von WebSphere MQ Classes for JMS-Anwendungen werden bestimmte Softwarekomponenten vorausgesetzt.

Aktuelle Informationen zu den Voraussetzungen für WebSphere MQ Classes for JMS finden Sie in der Readme-Datei von WebSphere MQ.

Für die Entwicklung von WebSphere MQ -Klassen für JMS-Anwendungen benötigen Sie ein Java 2 Software Development Kit (SDK). Auf der Seite mit den Systemvoraussetzungen für WebSphere MQ finden Sie Details zu den JDKs, die für Ihr Betriebssystem unterstützt werden. Siehe Voraussetzungen für WebSphere MQ.

Für die Ausführung von WebSphere MQ Classes for JMS-Anwendungen benötigen Sie die folgenden Softwarekomponenten:

- Einen WebSphere MQ-Warteschlangenmanager
- Eine Java Runtime Environment (JRE) für jedes System, auf dem Sie Anwendungen ausführen

Wenn Sie SSL-Verbindungen zur Verwendung von Verschlüsselungsmodulen benötigen, die FIPS 140-2-zertifiziert sind, benötigen Sie den IBM Java JSSE FIPS-Provider (IBMJSSEFIPS). Jedes IBM Java 2 SDK und jede JRE ab Version 5 enthält IBMJSSEFIPS.

Sie können Adressen für Internet Protocol Version 6 (IPv6) in Ihren WebSphere MQ -Klassen für JMS-Anwendungen verwenden, sofern IPv6 -Adressen von Ihrer Java Virtual Machine (JVM) und der TCP/IP-Implementierung auf Ihrem Betriebssystem unterstützt werden. Das WebSphere MQ-JMS-Verwaltungstool (siehe „Einsatz des WebSphere MQ JMS-Verwaltungstools” auf Seite 992) akzeptiert auch IPv6-Adressen.

Das JMS-Verwaltungstool WebSphere MQ und WebSphere MQ Explorer verwenden JNDI (Java Naming and Directory Interface), um auf einen Verzeichnisservice zuzugreifen, in dem verwaltete Objekte gespeichert werden. WebSphere MQ Classes for JMS-Anwendungen können auch JNDI verwenden, um verwaltete Objekte aus einem Verzeichnisservice abzurufen. Ein Service-Provider ist Code, der den Zugriff auf einen Verzeichnisservice bereitstellt, indem JNDI-Aufrufe den Aufrufen an den Verzeichnisservice zugeordnet werden. Mit WebSphere MQ Classes for JMS werden die folgenden Service-Provider bereitgestellt:

- Ein LDAP-Service-Provider (LDAP = Lightweight Directory Access Protocol) in den Dateien ldap.jar und providerutil.jar. Der LDAP-Service-Provider ermöglicht den Zugriff auf einen Verzeichnisservice auf Basis eines LDAP-Servers.
- Ein Dateisystem-Service-Provider in den Dateien fscontext.jar und providerutil.jar. 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 für das Speichern von Java-Objekten konfigurieren. In der Dokumentation des Servers finden Sie Informationen zur Installation und Konfiguration Ihres LDAP-Servers.

JMS-Programme für den IBM WebSphere MQ -Client für HP Integrity NonStop Server vorbereiten

In diesem Abschnitt finden Sie alle wichtigen Informationen, die Sie vor der Entwicklung und Ausführung von JMS-Programmen für den IBM WebSphere MQ-Client für HP Integrity NonStop Server benötigen.

Die IBM WebSphere MQ-Klassen für Java Message Service werden als Teil der Installation des IBM WebSphere MQ-Clients für HP Integrity NonStop Server installiert. Angaben zur Übersicht der Installationsinhalte finden Sie im Abschnitt Dateisystem.

Einige Aspekte der Clientfunktionalität sind hostbetriebssystemspezifisch. Weitere Informationen zu den unterstützten Funktionen für den IBM WebSphere MQ-Client für HP Integrity NonStop Server finden Sie im Abschnitt IBM WebSphere MQ-Client für HP Integrity NonStop Server-unterstützte Umgebungen und Funktionen.

Voraussetzungen

Um JMS-Anwendungen erstellen und ausführen zu können, muss die Komponente *HP Integrity NonStop Server for Java* installiert und verfügbar sein.

Einrichtung

Informationen zur Einrichtung der Umgebung für die Ausführung und Erstellung von Anwendungen, in denen IBM WebSphere MQ-Klassen für Java Message Service verwendet werden können, finden Sie im Abschnitt „Umgebungsvariablen, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden" auf Seite 766.

Informationen zu den erforderlichen Schritten zur Konfiguration eines Warteschlangenmanagers, sodass dieser Verbindungen von Clientanwendungen annimmt, finden Sie im Abschnitt „Konfiguration von Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, im Anschluss an die Installation" auf Seite 821.

Informationen zur Überprüfung Ihrer Umgebung mit IBM WebSphere MQ-Klassen für Java Message Service finden Sie im Abschnitt „Der Punkt-zu-Punkt-Installationsprüftest für WebSphere MQ Classes for JMS" auf Seite 824.

Anwendungen schreiben

Weitere Informationen zum Schreiben von JMS-Anwendungen finden Sie im Abschnitt „Anwendungen erstellen, die die WebSphere MQ-Klassen für JMS verwenden" auf Seite 856.

Weitere Informationen zur Verwendung des IBM WebSphere MQ-JMS-Verwaltungstools finden Sie im Abschnitt „Einsatz des WebSphere MQ JMS-Verwaltungstools" auf Seite 992.

Beispielanwendungen

Im folgenden Unterverzeichnis der Installation werden Beispielanwendungen bereitgestellt: `opt/mqm/samp/jms`.

Weitere Informationen zu den erforderlichen Konfigurationsschritten zur Ausführung der Beispiele finden Sie im Abschnitt „Beispielprogramme vorbereiten und ausführen" auf Seite 115.

Problemlösung

Informationen zur Behebung von Fehlern finden Sie im Abschnitt „Probleme in Zusammenhang mit den IBM WebSphere MQ-Klassen für JMS beheben" auf Seite 847.

Installation und Konfiguration von WebSphere MQ Classes for JMS

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

Zugehörige Konzepte

„Was ist für IBM WebSphere MQ Classes for JMS installiert?" auf Seite 764
Eine Reihe von Dateien und Verzeichnissen wird erstellt, wenn Sie IBM WebSphere MQ Classes for JMS installieren. 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 Umgebungsvariablen festlegen, damit Sie IBM WebSphere MQ -Klassen für JMS-Anwendungen ausführen können.

„ WebSphere MQ Classes for JMS-Anwendungen unter dem Sicherheitsmanager von Java ausführen" auf Seite 773
WebSphere MQ Classes for JMS kann mit aktiviertem Java -Sicherheitsmanager ausgeführt werden. Damit Anwendungen mit aktiviertem Sicherheitsmanager erfolgreich ausgeführt werden können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtliniendefinitionsdatei konfigurieren.

„Der IBM WebSphere MQ-Ressourcenadapter" auf Seite 777
Der Ressourcenadapter ermöglicht Anwendungen, die auf einem Anwendungsserver ausgeführt werden, den Zugriff auf IBM WebSphere MQ-Ressourcen. Er unterstützt sowohl die eingehende Kommunikation als auch die abgehende Kommunikation.

„Konfiguration von Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, im Anschluss an die Installation" auf Seite 821
In diesem Abschnitt erfahren Sie, welche Berechtigungen für Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, erforderlich sind, 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.

„Der Punkt-zu-Punkt-Installationsprüftest für WebSphere MQ Classes for JMS" auf Seite 824
Mit WebSphere 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.

„Der Publish/Subscribe-Installationsprüftest für WebSphere MQ Classes for JMS" auf Seite 828
Mit WebSphere 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.

„Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter" auf Seite 832
Das Programm des Installationsprüftests (Installation Verification Test, IVT) wird als EAR-Datei bereitgestellt. Um das Programm verwenden zu können, müssen Sie es implementieren und einige Objekte als JCA-Ressourcen definieren.

„Ressourcenadapter für abgehende Kommunikation konfigurieren" auf Seite 798
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

„Unterstützung von OSGi" auf Seite 846
OSGi stellt ein Framework bereit, das die Implementierung von Anwendungen als Bundles unterstützt. Mit den IBM WebSphere MQ classes for JMS werden neun OSGi-Bundles bereitgestellt.

„Probleme in Zusammenhang mit den IBM WebSphere MQ-Klassen für JMS beheben" auf Seite 847
Sie können Probleme untersuchen, indem Sie die Installationsprüfprogramme ausführen sowie Trace- und Protokollfunktionen nutzen.

Zugehörige Tasks

„MQ-Ressourcenadapter unter WAS CE installieren und testen" auf Seite 835
Installation des IBM WebSphere MQ-Ressourcenadapters und Ausführung der IVT-Anwendung (IVT = Installation Verification Test, Installationsprüftest) in WebSphere Application Server CE.

„IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren" auf Seite 837
Wenn Sie eine andere Warteschlange, einen anderen Warteschlangenmanager, einen anderen Port, einen anderen Host, einen anderen Kanal oder den Bindungsmodus anstelle des Clientmodus verwenden möchten, müssen Sie die IVT-Anwendung und die zugehörigen Scripts in WebSphere Application Server CE ändern, bevor Sie den Ressourcenadapter oder die IVT-Anwendung implementieren.

„IVT-Anwendung in JBoss mit einer angepassten IBM WebSphere MQ -Umgebung implementieren" auf Seite 840
Wenn Sie bei der Installation des IBM WebSphere MQ -Ressourcenadapters in JBosseine andere Warteschlange, einen anderen Warteschlangenmanager, einen anderen Port, einen anderen Host, einen anderen Kanal oder den Bindungsmodus anstelle des Clientmodus verwenden wollen, müssen Sie zuerst die IVT-Anwendung und die zugehörigen Scripts in JBoss ändern, bevor Sie den Ressourcenadapter oder die IVT-Anwendung implementieren.

Problembestimmung für den IBM WebSphere MQ-Ressourcenadapter

Zugehörige Verweise

„In den WebSphere MQ-Klassen für JMS bereitgestellte Scripts" auf Seite 845

Es werden mehrere Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die bei der Verwendung von WebSphere MQ-Klassen für JMS ausgeführt werden müssen.

Was ist für IBM WebSphere MQ Classes for JMS installiert?

Eine Reihe von Dateien und Verzeichnissen wird erstellt, wenn Sie IBM WebSphere MQ Classes for JMS installieren. 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 Umgebungsvariablen festlegen, damit Sie IBM WebSphere MQ -Klassen für JMS-Anwendungen ausführen können.

Bei den meisten Betriebssystemen werden die IBM WebSphere MQ -Klassen für JMS als optionale Komponente installiert, wenn Sie IBM WebSphere MQinstallieren. Für den IBM WebSphere MQ -Client für HP Integrity NonStop Serverwerden die IBM WebSphere MQ -Klassen für JMS standardmäßig installiert. Weitere Informationen zur Installation von IBM WebSphere MQfinden Sie unter:

WebSphere MQ-Server installieren
IBM WebSphere MQ-Client installieren

Tabelle 90 auf Seite 764 zeigt, wo die Dateien von IBM WebSphere MQ Classes for JMS auf den einzelnen Plattformen installiert sind.

Tabelle 90. Installationsverzeichnisse für IBM WebSphere MQ Classes for JMS

Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/java
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/java
HP-UX, Linuxund Solaris	*MQ_INSTALLATION_PATH*/java
Windows	*MQ_INSTALLATION_PATH*\java
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.	

Inhalt des Installationsverzeichnisses:

- Die JAR-Dateien von IBM WebSphere MQ Classes for JMS, die sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib befinden.
- Die nativen IBM WebSphere MQ -Bibliotheken, die von Anwendungen verwendet werden, die die native Java-Schnittstelle verwenden.

  Die nativen 32-Bit-Bibliotheken werden im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib installiert und die nativen 64-Bit-Bibliotheken befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib64 .

  Weitere Informationen zu den nativen IBM WebSphere MQ-Bibliotheken finden Sie im Abschnitt „JNI-Bibliotheken (Java Native Interface) konfigurieren” auf Seite 768.
- Zusätzliche Scripts, die in „In den WebSphere MQ-Klassen für JMS bereitgestellte Scripts” auf Seite 845 beschrieben sind. Diese Scripts befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\bin.
- Die Spezifikationen der API von IBM WebSphere 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.

  Auf UNIX-, Linux-und Windows-Systemen enthält dieses Unterverzeichnis die einzelnen HTML-Seiten.
- Unterstützung von OGSi. OSGi-Bundles werden im Verzeichnis java\lib\OSGi installiert und im Abschnitt „Unterstützung von OSGi” auf Seite 846 beschrieben.
- Der IBM WebSphere MQ -Ressourcenadapter, der in jedem mit JCA 1.5 (oder höher) kompatiblen Anwendungsserver implementiert werden kann

Der IBM WebSphere MQ -Ressourcenadapter befindet sich im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\jca. Weitere Informationen finden Sie unter „Der IBM WebSphere MQ-Ressourcenadapter” auf Seite 777

- Unter Windows werden Symbole, die für das Debugging verwendet werden können, im Verzeichnis "*MQ_INSTALLATION_PATH*\java\lib\symbols" installiert.

Darüber hinaus enthält das Installationsverzeichnis einige Dateien, die zu sonstigen IBM WebSphere MQ-Komponenten gehören. Dabei handelt es sich um die folgenden Verzeichnisse:

- Der IBM WebSphere MQ -Transport für SOAP, der einen JMS-Transport für SOAP bereitstellt, wird im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\soap installiert. Weitere Informationen zu IBM WebSphere MQ Transport for SOAP finden Sie im Abschnitt „WebSphere MQ-Transport für SOAP” auf Seite 1005im Information Center.
- Auf verteilten Plattformen ist IBM WebSphere MQ Bridge for HTTP im Verzeichnis *MQ_INSTALLATION_PATH*\java\lib\http installiert. Weitere Informationen zu IBM WebSphere MQ Bridge for HTTP finden Sie im Abschnitt des Infocenters, in dem „WebSphere MQ Bridge for HTTP” auf Seite 1084 beschrieben wird.

Einige Beispielanwendungen werden mit IBM WebSphere MQ Classes for JMS bereitgestellt. In Tabelle 91 auf Seite 765 sehen Sie, wo die Beispielanwendungen auf den einzelnen Plattformen installiert werden.

Tabelle 91. Beispielverzeichnisse

Plattform	**Directory**
AIX	*MQ_INSTALLATION_PATH*/samp/jms
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/samp/jms
HP-UX, Linuxund Solaris	*MQ_INSTALLATION_PATH*/samp/jms
Windows	*MQ_INSTALLATION_PATH*\tools\jms
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.	

Nach der Installation müssen Sie möglicherweise einige Konfigurationsaufgaben ausführen, um Anwendungen zu kompilieren und auszuführen.

„Umgebungsvariablen, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden” auf Seite 766 beschreibt den Klassenpfad, der für die Ausführung einfacher IBM WebSphere MQ -Klassen für JMS-Anwendungen erforderlich ist. In diesem Artikel werden auch zusätzliche JAR-Dateien beschrieben, die unter bestimmten Umständen referenziert werden müssen, sowie die Umgebungsvariablen, die Sie festlegen müssen, um die Scripts auszuführen, die mit IBM WebSphere MQ Classes for JMS bereitgestellt werden.

Wenn Ihre IBM WebSphere MQ Classes for JMS-Anwendung eine Verbindung zu Code herstellen soll, der in anderen Sprachen als Java geschrieben ist (z. B. zur Verwendung des Bindungstransports beim Herstellen einer Verbindung zu einem Warteschlangenmanager), erklärt „JNI-Bibliotheken (Java Native Interface) konfigurieren” auf Seite 768 , wo sich die Position der JNI-Bibliotheken (Java Native Interface) befindet, die als Parameter des Java-Befehls angegeben werden müssen.

Zur Steuerung von Eigenschaften wie die Traceverarbeitung und Protokollierung bei einer Anwendung müssen Sie eine Konfigurationseigenschaftendatei bereitstellen. Die Konfigurationseigenschaftendatei für IBM WebSphere MQ Classes for JMS wird in „Die Konfigurationsdatei IBM WebSphere MQ classes for JMS” auf Seite 771beschrieben.

Installation und Upgrade der JAR-Dateien von WebSphere MQ Classes for JMS

Die einzige unterstützte Methode zum Abrufen der JAR-Dateien von IBM WebSphere MQ Classes for JMS auf einem System besteht darin, entweder das Produkt IBM WebSphere MQ oder das WebSphere MQ V7.5 Clients SupportPac- MQC75zu installieren oder ein Software-Management-Tool wie Apache

Maven zu verwenden. Weitere Informationen finden Sie im Abschnitt „IBM WebSphere MQ classes for JMS und Software-Management-Tools” auf Seite 772.

Verschieben oder kopieren Sie die JAR-Dateien oder nativen Bibliotheken von IBM WebSphere MQ Classes for JMS nicht auf andere Maschinen oder an eine andere Position auf einer Maschine, auf der IBM WebSphere MQ Classes for JMS installiert wurde, es sei denn, Sie verwenden ein Software-Management-Tool.

- Auf eine "Installation", bei der JAR-Dateien von einer anderen Maschine kopiert wurden, können keine Fixpacks angewendet werden, da es dabei viel schwieriger ist sicherzustellen, dass alle JAR-Dateien miteinander Schritt halten und in kompatiblen Versionen vorliegen.
- Das Kopieren der JAR-Dateien von IBM WebSphere MQ Classes for JMS zwischen Maschinen kann auch dazu führen, dass sich mehrere Kopien der Dateien auf derselben Maschine befinden, was zu Problemen bei der Wartung des Codes und beim Debugging führen kann.
- Mit dem Befehl dspmqver , mit dem Versionsinformationen aus einer IBM WebSphere MQ -Installation angezeigt werden, werden nur Versionsinformationen für die im Verzeichnis `\java\lib` installierten IBM WebSphere MQ Classes for JMS angezeigt.

  Wenn sich mehrere Kopien der Dateien auf derselben Maschine befinden, liefert die Ausführung von **dspmqver** möglicherweise keine genauen Informationen zur Version der IBM WebSphere MQ -Klassen für JMS, die von einer Anwendung verwendet werden.

Schließen Sie die JAR-Dateien von IBM WebSphere MQ Classes for JMS nicht in Anwendungsarchive (z. B. Unternehmensanwendungsarchive oder EAR-Dateien) ein.

- Aktualisierungen der IBM WebSphere MQ -Klassen für JMS können nicht mit einem IBM WebSphere MQ -Fixpack angewendet werden.
- Der IBM Support kann die Version der IBM WebSphere MQ -Klassen für JMS, die von der Anwendung verwendet werden, nicht ohne großen Aufwand ermitteln.
- Probleme können auftreten, wenn mehrere Anwendungen, die in derselben Java Runtime Environment ausgeführt werden, unterschiedliche Versionen der IBM WebSphere MQ -Klassen für JMS enthalten, da mehrere Versionen der IBM WebSphere MQ -Klassen für JMS gleichzeitig in die Java Runtime Environment geladen werden.

  Beispiele für diese Probleme sind die folgenden Ausnahmebedingungen:

```
java.lang.ClassCastException :
com.ibm.mq.jmqi.system.JmqiSystemEnvironment incompatible with
com.ibm.mq.jmqi.system.JmqiSystemEnvironment

java.lang.ClassCastException :
com.ibm.mq.jms.MQQueue incompatible with com.ibm.mq.jms.MQQueue
```

- Wenn eine Anwendung den BINDINGS-Transport verwendet, um eine Verbindung zu einem Warteschlangenmanager herzustellen, müssen alle wichtigen Upgrades für den Warteschlangenmanager auch die Anwendung so aktualisieren, dass sie die entsprechende Version der IBM WebSphere MQ -Klassen für JMS enthält.

  Wenn beispielsweise für einen Warteschlangenmanager ein Upgrade auf IBM WebSphere MQ Version 7.5 durchgeführt wird, müssen auch alle Anwendungen, die über den BINDINGS-Transport eine Verbindung zum Warteschlangenmanager herstellen, so aktualisiert werden, dass sie die IBM WebSphere MQ Version 7.5 -Klassen für JMS enthalten.

Umgebungsvariablen, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden

Bevor Sie IBM WebSphere MQ Classes for JMS-Anwendungen kompilieren und ausführen können, muss die Einstellung für die Umgebungsvariable CLASSPATH die JAR-Datei (Java Archive) von IBM WebSphere MQ Classes for JMS enthalten. Abhängig von den gegebenen Voraussetzungen müssen Sie Ihrem Klassenpfad möglicherweise weitere JAR-Dateien hinzufügen. Um die von den IBM WebSphere MQ-Klassen für JMS bereitgestellten Scripts auszuführen, müssen weitere Umgebungsvariablen gesetzt werden.

Damit Anwendungen, die die IBM WebSphere MQ-Klassen für JMS verwenden, kompiliert und ausgeführt werden können, müssen Sie die in Tabelle 92 auf Seite 767 für Ihre Plattform aufgeführte Einstellung für die Umgebungsvariable CLASSPATH verwenden. Die Einstellung enthält das Beispielverzeichnis, damit Sie Beispielanwendungen, die die IBM WebSphere MQ-Klassen für JMS verwenden, kompilieren und ausführen können. Alternativ zur Verwendung der Umgebungsvariablen können Sie den Klassenpfad auch im **java**-Befehl angeben.

Tabelle 92. CLASSPATH-Einstellung für die Kompilierung und Ausführung von Anwendungen (einschließlich der Beispielanwendungen), die die IBM WebSphere MQ-Klassen für JMS verwenden

Plattform	**CLASSPATH-Einstellung**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP Integrity Non-Stop Server	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP-UX, Linuxund Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms;
z/OS	CLASSPATH=*MQ_INSTALLATION_PATH*/mqm/V7R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V6R0M0/java/samples/jms:
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.	

Das Manifest für die JAR-Datei 'com.ibm.mqjms.jar' enthält Verweise auf die meisten anderen JAR-Dateien, die von Anwendungen, die die IBM WebSphere MQ-Klassen für JMS verwenden, erforderlich sind, daher müssen diese JAR-Dateien dem Klassenpfad nicht hinzugefügt werden. Diese JAR-Dateien enthalten die Dateien, die von Anwendungen benötigt werden, die JNDI (Java Naming and Directory Interface) verwenden, um verwaltete Objekte aus einem Verzeichnisservice abzurufen, und von Anwendungen, die JTA (Java Transaction API) verwenden.

In folgenden Situationen müssen Sie jedoch zusätzliche JAR-Dateien in Ihren Klassenpfad aufnehmen:

- Wenn Sie Kanalexitklassen verwenden, die die im Paket com.ibm.mq definierten Kanalexitschnittstellen anstelle der im Paket com.ibm.mq.exits definierten implementieren, müssen Sie die JAR-Datei com.ibm.mq.jarvon IBM WebSphere MQ Classes for Java zu Ihrem Klassenpfad hinzufügen.
- Wenn Sie Ihren Java-Code mit einem Java 2 Software Development Kit (SDK) der Version 1.4.2kompilieren, müssen Sie Ihrem Klassenpfad die folgenden JAR-Dateien hinzufügen:
  - jms.jar
  - com.ibm.mq.jmqi.jar

  Wenn Ihre Anwendungen außerdem verwaltete Objekte über JNDI aus einem Verzeichnisservice abrufen, müssen Sie darüber hinaus die folgenden JAR-Dateien in Ihren Klassenpfad aufnehmen:
  - fscontext.jar
  - jndi.jar

– ldap.jar
– providerutil.jar

Wenn Ihre Anwendung die Java Transaction API nutzt, müssen Sie außerdem die Datei 'jta.jar' in Ihren Klassenpfad aufnehmen.

Beachten Sie, dass diese zusätzlichen JAR-Dateien nur für die Kompilierung Ihrer Anwendungen erforderlich sind, nicht für deren Ausführung.

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 Extended Security Edition installiert haben.

Die von den IBM WebSphere MQ-Klassen für 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 denen die WebSphere MQ-Klassen für JMS installiert sind.

MQ_JAVA_LIB_PATH
Diese Umgebungsvariable gibt das Verzeichnis an, in dem die Bibliotheken der WebSphere MQ-Klassen für JMS gespeichert sind; siehe Tabelle 93 auf Seite 769.

Unter Windows werden alle Umgebungsvariablen automatisch während der Installation festgelegt. Auf allen anderen Plattformen müssen Sie sie selbst einstellen.

Zur Einrichtung der Umgebungsvariablen auf einer 32-Bit-JVM unter UNIX, HP Integrity NonStop Server oder Linux können Sie das Script 'setjmsenv' verwenden. Um bei Verwendung einer 64-Bit-JVM auf einem UNIX oder Linux-System die Umgebungsvariablen festzulegen, können Sie das Script 'setjmsenv64' verwenden. Diese Scripts befinden sich im Verzeichnis *MQ_INSTALLATION_PATH*`/java/bin` , wobei *MQ_INSTALLATION_PATH* das übergeordnete Verzeichnis ist, in dem IBM WebSphere 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. Sie können das Script auch in jeder Sitzung ausführen, über die JMS-Startscripts ausgeführt werden sollen. Wenn Sie diese Option auswählen, müssen Sie das Script in jedem Shellfenster ausführen, das Sie während des JMS-Prüfprozesses starten, indem Sie `. ./setjmsenv` oder `. ./setjmsenv64` eingeben.

JNI-Bibliotheken (Java Native Interface) konfigurieren

IBM WebSphere MQ Classes for JMS-Anwendungen, die entweder über den Bindungstransport eine Verbindung zu einem Warteschlangenmanager herstellen oder über den Clienttransport eine Verbindung zu einem Warteschlangenmanager herstellen und die Kanalexitprogramme verwenden, die in anderen Sprachen als Java geschrieben sind, müssen in einer Umgebung ausgeführt werden, die auf die JNI-Bibliotheken (JNI = Java Native Interface) zugreift.

Informationen zu diesem Vorgang

Zum Einrichten dieser Umgebung müssen Sie den Bibliothekspfad der Umgebung konfigurieren, damit die JVM (Java Virtual Machine) die Bibliothek 'mqjbnd' laden kann, bevor Sie die Anwendung ' IBM WebSphere MQ Classes for JMS' starten.

IBM WebSphere MQ stellt zwei JNI-Bibliotheken (Java Native Interface) bereit:

mqjbnd
Diese Bibliothek wird von Anwendungen verwendet, die mithilfe des Bindungstransports eine Verbindung zu einem Warteschlangenmanager herstellen. Sie stellt die Schnittstelle zwischen den IBM WebSphere MQ -Klassen für JMS und dem Warteschlangenmanager bereit. Die Bibliothek mqjbnd , die mit IBM WebSphere MQ Version 7.5 installiert wird, kann verwendet werden, um eine Verbindung

zu einem beliebigen Warteschlangenmanager der IBM WebSphere MQ Version 7.5 (oder früher) herzustellen.

mqjexitstub02
Die Bibliothek mqjexitstub02 wird von den IBM WebSphere MQ -Klassen für 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 WebSphere MQ 32-Bit- und 64-Bit-Versionen dieser JNI-Bibliotheken. Die Position der Bibliotheken bei den einzelnen Plattformen wird in Tabelle 1 aufgeführt.

Tabelle 93. Position der IBM WebSphere MQ Classes for JMS-Bibliotheken für jede Plattform

Plattform	**Verzeichnis mit den Bibliotheken der IBM WebSphere MQ -Klassen für JMS**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
HP-UX	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Linux (POTENZ , x86-64 und zSeries s390x -Plattformen)	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Linux (x86-Plattform) Linux (zSeries -Plattform)	*MQ_INSTALLATION_PATH*/java/lib
Solaris (x86-64- und SPARC-Plattformen)	*MQ_INSTALLATION_PATH*/java/lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*/java/lib64 (64-Bit-Bibliotheken)
Windows	*MQ_INSTALLATION_PATH*\java\lib (32-Bit-Bibliotheken) *MQ_INSTALLATION_PATH*\Java\lib64 (64-Bit-Bibliotheken)
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem IBM WebSphere MQ installiert ist.	

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 WebSphere MQ installiert haben. Für eine 64-Bit-JVM und eine IBM WebSphere MQ-Standardinstallationsposition können Sie beispielsweise folgende Einstellungen verwenden:

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

  Solaris HP-UX Linux
  ```
  export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIB
  RARY_PATH
  ```

  Windows
  ```
  set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
  ```

  Es folgt ein Beispiel für den Ausnahmebedingungsstack, den Sie sehen, wenn die Umgebung nicht ordnungsgemäß konfiguriert wurde:

```
Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Failed to load the WebSphere MQ native JNI library: 'mqjbnd'.
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    at java.security.AccessController.doPrivileged(AccessController.java:400)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.ja-
va:86)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccesso-
rImpl.java:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQCon-
nectionFactory.java:8437)
    ... 7 more
Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 more
```

2. Starten Sie nach der Konfiguration der 32 -Bit-oder 64-Bit-Umgebung die Anwendung IBM WebSphere MQ Classes for JMS mit dem folgenden Befehl:

   ```
   java application-name
   ```

   Dabei ist *anwendungsname* der Name der auszuführenden IBM WebSphere MQ Classes for JMS-Anwendung.

   Eine Ausnahmebedingung, die den IBM WebSphere MQ -Ursachencode 2495 (MQRC_MODULE_NOT_FOUND) enthält, wird von den IBM WebSphere MQ -Klassen für JMS ausgelöst, wenn:

   - Die Anwendung IBM WebSphere MQ Classes for JMS wird in einer 32-Bit-Java-Laufzeitumgebung ausgeführt und eine 64-Bit-Umgebung wurde für die IBM WebSphere MQ -Klassen für JMS konfiguriert, da die 32-Bit-Java-Laufzeitumgebung die native 64-Bit-Java-Bibliothek nicht laden kann.
   - Die Anwendung IBM WebSphere MQ Classes for JMS wird in einer 64-Bit-Java-Laufzeitumgebung ausgeführt und eine 32-Bit-Umgebung wurde für IBM WebSphere MQ Classes for JMS eingerichtet, da die 64-Bit-Java-Laufzeitumgebung die native 32-Bit-Java-Bibliothek nicht laden kann.

Die Konfigurationsdatei IBM WebSphere MQ classes for JMS

In einer Konfigurationsdatei von WebSphere MQ Classes for JMS sind bestimmte Eigenschaften angegeben, mit denen WebSphere MQ Classes for JMS konfiguriert wird.

Das Format einer Konfigurationsdatei für WebSphere MQ Classes for JMS ist das einer Java-Standardeigenschaftendatei. Das Unterverzeichnis `bin` des Installationsverzeichnisses von WebSphere MQ Classes for JMS enthält eine Beispielkonfigurationsdatei namens 'jms.config'. In dieser Datei sind alle unterstützten Eigenschaften und ihre Standardwerte dokumentiert.

Sie können den Namen und die Speicherposition einer Konfigurationsdatei von WebSphere 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 diesem Befehl steht *URL_der_Konfigurationsdatei* für eine URL, die den Namen und die Speicherposition der Konfigurationsdatei von WebSphere 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
```

Dieser Befehl gibt die Konfigurationsdatei von WebSphere MQ Classes for JMS als die Datei D:\mydir\mjms.config auf dem lokalen Windows-System an.

Beim Start einer Anwendung liest WebSphere MQ Classes for JMS den Inhalt der Konfigurationsdatei und speichert die angegebenen Eigenschaften in einem internen Eigenschaftenspeicher. Wenn eine Konfigurationsdatei nicht mit dem **java**-Befehl angegeben ist oder nicht gefunden werden kann, verwendet WebSphere MQ Classes for JMS 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 von WebSphere MQ Classes for JMS kann in Verbindung mit allen unterstützten Transportmethoden zwischen einer Anwendung und einem Warteschlangenmanager oder Broker genutzt werden.

Beachten Sie, dass Sie keinen Starttrace festlegen können, indem Sie in der Konfigurationsdatei von WebSphere MQ Classes for JMS eine entsprechende Eigenschaft festlegen. Die Angabe eines Starttrace ist nur durch die Festlegung einer Systemeigenschaft im **java**-Befehl möglich. Beispiel:

```
java -Dcom.ibm.msg.client.commonservices.trace.startup=true
     -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
      MyAppClass
```

In einer WebSphere MQ-Clientkonfigurationsdatei angegebene Eigenschaften überschreiben

Eine WebSphere MQ-Clientkonfigurationsdatei kann auch Eigenschaften angeben, die für die Konfiguration von WebSphere MQ Classes for JMS verwendet werden. Die in einer WebSphere MQ-Clientkonfigurationsdatei angegebenen Eigenschaften gelten jedoch nur, wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt.

Falls erforderlich, können Sie alle Attribute in einer WebSphere MQ-Clientkonfigurationsdatei überschreiben, indem Sie sie in einer Konfigurationsdatei für WebSphere MQ Classes for JMS als Eigenschaft angeben. Wenn Sie ein Attribut in einer WebSphere MQ-Clientkonfigurationsdatei überschreiben möchten, verwenden Sie in der Konfigurationsdatei für WebSphere 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 WebSphere MQ-Clientkonfigurationsdatei, die das Attribut enthält

propName
Der Name des Attributs, das in der WebSphere MQ-Clientkonfigurationsdatei angegeben ist

propValue
Der Wert der Eigenschaft, der den Wert des Attributs, das in der WebSphere MQ-Clientkonfigurationsdatei angegeben ist, überschreibt

Sie können ein Attribut in einer WebSphere MQ-Clientkonfigurationsdatei aber auch überschreiben, indem Sie die Eigenschaft als Systemeigenschaft im **java**-Befehl angeben. Verwenden Sie das vorherige Format zur Angabe der Eigenschaft als Systemeigenschaft.

Für WebSphere MQ Classes for JMS sind nur die folgenden Attribute in einer WebSphere MQ-Clientkonfigurationsdatei 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

IBM WebSphere MQ classes for JMS und Software-Management-Tools

Mit den IBM WebSphere 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 WebSphere MQ classes for JMS setzen sich aus einer Reihe von JAR-Dateien zusammen. Wenn Sie Java-Sprachanwendungen mithilfe dieser API entwickeln, ist eine Installation eines SupportPac von IBM WebSphere MQ Server, Client oder Client auf der Maschine erforderlich, auf der die Anwendung entwickelt wird.

Wenn Sie ein solches Tool verwenden und die JAR-Dateien, die die IBM WebSphere 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 WebSphere MQ-Release oder -Fixpack enthalten.

- Es ist Ihre Aufgabe, das Repository mit allen vom IBM Support zur Verfügung gestellten Wartungsreleases zu aktualisieren.

Bei IBM WebSphere MQ Version 7.5 müssen die folgenden JAR-Dateien im Repository installiert werden:

- com.ibm.mqjms.jar.
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- CL3Export.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- CL3Nonexport.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- jndi.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- ldap.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- rmm.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- dhbcore.jar ist bei Verwendung der IBM WebSphere MQ classes for JMS erforderlich.
- jms.jar ist erforderlich, wenn Sie die IBM WebSphere MQ classes for JMS verwenden.
- fscontext.jar ist erforderlich, wenn Sie die IBM WebSphere MQ classes for JMS verwenden und auf verwaltete JMS-Objekte zugreifen, die in einem Dateisystem-JNDI-Kontext gespeichert sind.
- providerutil.jar, wenn Sie die IBM WebSphere MQ classes for JMS verwenden und auf verwaltete JMS-Objekte zugreifen, die in einem Dateisystem-JNDI-Kontext gespeichert sind.

WebSphere MQ Classes for JMS-Anwendungen unter dem Sicherheitsmanager von Java ausführen

WebSphere MQ Classes for JMS kann mit aktiviertem Java -Sicherheitsmanager ausgeführt werden. Damit Anwendungen mit aktiviertem Sicherheitsmanager erfolgreich ausgeführt werden können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtliniendefinitionsdatei konfigurieren.

Die einfachste Methode besteht darin, die zusammen mit der JRE bereitgestellte Richtlinienkonfigurationsdatei zu ändern. Auf den meisten Systemen ist diese Datei im Pfad `lib/security/java.policy` (relativ zum JRE-Verzeichnis) gespeichert. Sie können die Richtlinienkonfigurationsdatei mit Ihrem bevorzugten Editor oder mit dem Programm `policytool` bearbeiten, das mit Ihrer JRE bereitgestellt wird.

Wichtig: V 7.5.0.8 Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Eine Ausnahme bilden die folgenden Java-Systemeigenschaftsnamen.

Wenn Sie den Sicherheitsmanagermechanismus Java mit Ihrer Anwendung verwenden, müssen Sie die folgenden 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.whitelist, com.ibm.mq.jms.whitelist.discover und com.ibm.mq.jms.whitelist.mode.

Die Zulassungsliste ClassName wird mit APAR IT14385 und IBM WebSphere MQ Version 7.5.0Fixpack 8unterstützt. Weitere Informationen finden Sie unter „ClassName -Zulassungsliste in JMS ObjectMessage” auf Seite 774.

Es folgen zwei Beispieleinträge in einer Richtlinienkonfigurationsdatei, die die erfolgreiche Ausführung von WebSphere MQ-Klassen für JMS unter dem Standardsicherheitsmanager ermöglichen:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
```

```
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //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
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "console.encoding","read";
  permission java.lang.RuntimePermission "setContextClassLoader";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.*","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
  permission java.util.logging.LoggingPermission "control";
  //Wherever trace output is expected
  permission java.io.FilePermission "/tmp/*","read,write";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
```

Anmerkungen:

- *MQ_INSTALLATION_PATH* steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.
- Die erste `grant`-Anweisung enthält die für die WebSphere MQ-Klassen für JMS erforderlichen Berechtigungen, die zweite `grant`-Anweisung enthält die Berechtigungen, die für eine Anwendung erforderlich sind, die die WebSphere MQ-Klassen für JMS verwendet.
- Fügen Sie der ersten Anweisung `grant` die folgende Berechtigung hinzu, damit WebSphere MQ Classes for JMS auf die Java -Archivdateien (JAR-Dateien) einer Anwendung zugreifen kann:

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- Damit diese `grant`-Anweisungen in der Richtlinienkonfigurationsdatei verwendet werden können, müssen Sie die Pfadnamen unter Umständen in die Pfade ändern, in denen die WebSphere MQ-Klassen für JMS installiert und Ihre Anwendungen gespeichert sind.
- Die mit den WebSphere MQ-Klassen für JMS bereitgestellten Beispielanwendungen und die Scripts zu ihrer Ausführung aktivieren den Sicherheitsmanager nicht.

V 7.5.0.8 ClassName -Zulassungsliste in JMS ObjectMessage

V 7.5.0.8 In WebSphere MQ Classes for JMS bietet die Unterstützung für die Zulassungsliste von Klassen in der Implementierung der JMS-Schnittstelle ObjectMessage eine potenzielle Risikominderung gegen einige der Sicherheitsrisiken, die sich potenziell auf den Serialisierungs-und Deserialisierungsmechanismus für Java-Objekte beziehen.

Anmerkung: Soweit dies möglich ist, wird der Begriff *allowlist* durch den Begriff *whitelist* ersetzt. Davon ausgenommen sind die Java-Systemeigenschaftsnamen, die in diesem Thema genannt werden.

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 wichtige Anwendung der Serialisierung ist in ObjectMessages von Java Message Service (JMS) zu finden, die die Serialisierung verwenden, um beliebige Objekte zu kapseln und zu übertragen.

Eine Zulassungsliste für die Serialisierung stellt eine Möglichkeit dar, einige der durch die Serialisierung entstehenden Risiken zu mindern. Indem explizit angegeben wird, welche Klassen in ObjectMessages eingebunden und welche daraus extrahiert werden können, bietet die Verwendung einer Zulassungsliste einen gewissen Schutz vor einigen Serialisierungsrisiken.

Zulassungsliste in WebSphere MQ Classes for JMS

Mit APAR IT14385 und IBM WebSphere MQ Version 7.5.0Fixpack 8unterstützt WebSphere MQ Classes for JMS die Zulassungsliste von Klassen in der Implementierung der JMS-Schnittstelle ObjectMessage . 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: WebSphere MQ Classes for JMS kann nicht mit einer Zulassungsliste verteilt werden. Die Auswahl von Klassen, die mithilfe von ObjectMessage übertragen werden sollen, erfolgt im Anwendungsdesign und kann nicht von IBM WebSphere 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.whitelist
: 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 -Sicherheitsmanager verwenden, müssen Sie sicherstellen, dass die JAR-Dateien von WebSphere MQ Classes for JMS Lese-und Schreibzugriff auf diese Datei haben.

com.ibm.mq.jms.whitelist.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.whitelist angegeben wurde, ist 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.whitelist.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 befindet sich in der Standardplattformdateicodierung mit plattformspezifischen Zeilenenden.

  Anmerkung: Wenn Sie zwischen heterogenen Systemen hin- und herwechseln, muss die Datei unter Umständen konvertiert werden.
- 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 als durch Kommas getrennte Liste von Klassennamen angegeben wird, wird sie unter der Annahme, dass das Betriebssystem oder die Shell sie nicht verarbeitet, als Standardkommentarzeichen in vielen UNIX -oder Linux -Shells als normales Zeichen behandelt.

Wann kommt der Zulassungslistenmechanismus zum Einsatz?

Der Zulassungslistenmechanismus wird initialisiert, wenn die Anwendung das erste Mal die ObjectMessage-Methode setMessage() oder getMessage() ausführt.

Die Systemeigenschaften werden ausgewertet, die Zulassungslistendatei wird geöffnet und in der Betriebsart ENFORCEMENT wird bei der Initialisierung des Mechanismus die Liste der in der Zulassungsliste aufgeführten Klassen geladen. An diesem Punkt wird ein Eintrag in die IBM WebSphere 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 IBM WebSphere MQ Advanced Message Security-(AMS-)Mechanismen kann die Sicherheit erhöht werden, indem sichergestellt wird, dass Nachrichten aus vertrauenswürdigen Quellen stammen

Wenn Sie den Sicherheitsmanagermechanismus Java mit Ihrer Anwendung verwenden, müssen Sie die folgenden 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.whitelist, com.ibm.mq.jms.whitelist.discover und com.ibm.mq.jms.whitelist.mode.

Zugehörige Konzepte

„ WebSphere MQ Classes for JMS-Anwendungen unter dem Sicherheitsmanager von Java ausführen” auf Seite 773

WebSphere MQ Classes for JMS kann mit aktiviertem Java -Sicherheitsmanager ausgeführt werden. Damit Anwendungen mit aktiviertem Sicherheitsmanager erfolgreich ausgeführt werden können, müssen Sie Ihre Java Virtual Machine (JVM) mit einer geeigneten Richtliniendefinitionsdatei konfigurieren.

Der IBM WebSphere MQ-Ressourcenadapter

Der Ressourcenadapter ermöglicht Anwendungen, die auf einem Anwendungsserver ausgeführt werden, den Zugriff auf IBM WebSphere MQ-Ressourcen. Er unterstützt sowohl die eingehende Kommunikation als auch die abgehende Kommunikation.

Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) bietet eine Standardmethode, Anwendungen, die in einer Java EE -Umgebung ausgeführt werden, mit einem unternehmensweiten Informationssystem (EIS) wie IBM WebSphere MQ oder Db2zu verbinden. Der IBM WebSphere MQ-Ressourcenadapter implementiert die JCA-Schnittstellen der Version 1.5 und enthält die IBM WebSphere 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 WebSphere MQ-Warteschlangenmanagers. Der Ressourcenadapter unterstützt sowohl die Punkt-zu-Punkt-Domäne als auch die Publish/Subscribe-Domäne.

Der IBM WebSphere 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.

Weitere Informationen zu IBM WebSphere MQ classes for JMS finden Sie im Abschnitt „WebSphere MQ-Klassen für JMS verwenden" auf Seite 759.

Der Ressourcenadapter enthält auch die IBM WebSphere MQ classes for Java. Die Klassen sind automatisch für Anwendungen verfügbar, die in einem Anwendungsserver ausgeführt werden, in dem der Ressourcenadapter implementiert wurde, und ermöglichen Anwendungen, die in diesem Anwendungsserver ausgeführt werden, die Verwendung der IBM WebSphere MQ classes for Java -API beim Zugriff auf Ressourcen eines IBM WebSphere MQ -Warteschlangenmanagers. Weitere Informationen zu IBM WebSphere MQ classes for Java finden Sie im Abschnitt „WebSphere MQ Classes for Java verwenden" auf Seite 693.

Die Verwendung von IBM WebSphere MQ classes for Java in einer Java EE -Umgebung wird mit Einschränkungen unterstützt. Informationen zu diesen Einschränkungen finden Sie im Abschnitt „Anwendungen, die IBM WebSphere MQ-Klassen für Java verwenden, in Java Platform, Enterprise Edition ausführen" auf Seite 757.

Weitere erforderliche Dokumentation für die Unterstützung eines JCA-Ressourcenadapters

In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Konfiguration eines JCA-Ressourcenadapters.

Jeder Anwendungsserver stellt eine eigene Gruppe von Verwaltungsschnittstellen zur Verfügung. Manche 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 WebSphere MQ-Ressourcenadapters bereitzustellen. Diese Dokumentation konzentriert sich nur auf das, was Sie konfigurieren müssen. In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Konfiguration eines JCA-Ressourcenadapters.

Damit diese Dokumentation verständlich ist, müssen Sie mit JMS und WebSphere MQ Classes for JMS vertraut sein. Viele der Eigenschaften, mit denen der WebSphere MQ-Ressourcenadapter konfiguriert wird, sind äquivalent zu den Eigenschaften der WebSphere MQ Classes for JMS-Objekte und weisen dieselben Funktionen auf.

Installation des WebSphere MQ-Ressourcenadapters

Der WebSphere MQ-Ressourcenadapter wird als RAR-Datei (Ressourcenarchivdatei) bereitgestellt. Installieren Sie die RAR-Datei auf Ihrem Anwendungsserver. Möglicherweise müssen Sie dem Systempfad Verzeichnisse hinzufügen.

Der WebSphere MQ-Ressourcenadapter wird als RAR-Datei (Ressourcenarchivdatei) mit dem Namen 'wmq.jmsra.rar' bereitgestellt. Diese Datei wird mit WebSphere MQ Classes for JMS in dem Verzeichnis installiert, das in Tabelle 94 auf Seite 778 aufgelistet ist.

Tabelle 94. Verzeichnis mit der Datei 'wmq.jmsra.rar' auf den einzelnen Plattformen

Plattform	**Directory**
AIX, HP-UX, Linux und Solaris	*MQ_INSTALLATION_PATH*/java/lib/jca

Tabelle 94. Verzeichnis mit der Datei 'wmq.jmsra.rar' auf den einzelnen Plattformen (Forts.)	
Plattform	**Directory**
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
MQ_INSTALLATION_PATH steht für das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.	

Die RAR-Datei enthält WebSphere MQ Classes for JMS und die WebSphere MQ-Implementierungen der JCA-Schnittstellen.

Sie müssen die RAR-Datei des WebSphere MQ-Ressourcenadapters auf Ihrem Anwendungsserver installieren. Die Installationsweise hängt vom jeweiligen Anwendungsserver ab. In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Installation einer RAR-Datei des Ressourcenadapters.

Für Bindungsverbindungen auf UNIX and Linux -Systemen müssen Sie sicherstellen, dass sich das Verzeichnis mit den JNI-Bibliotheken (JNI = Java Native Interface) im Systempfad befindet. Sie finden Angaben zur Position dieses Verzeichnisses, das auch die Bibliotheken von WebSphere MQ Classes for JMS enthält, in Tabelle 93 auf Seite 769. Unter Windows wird dieses Verzeichnis automatisch während der Installation von WebSphere MQ Classes for JMS zum Systempfad hinzugefügt.

Transaktionen werden sowohl im Clientmodus als auch im Bindungsmodus unterstützt.

Der WebSphere MQ-Ressourcenadapter und die Version von WebSphere MQ Classes for JMS, die durch den Ressourcenadapter verwendet wird, müssen auf demselben Release-Level sein.

WebSphere Application Server und der WebSphere MQ-Ressourcenadapter
Verwenden Sie den Ressourcenadapter WebSphere MQ nicht mit WebSphere Application Server Version 6. WebSphere Application Server V7 enthält eine Version des WebSphere MQ-Ressourcenadapters der Version 7.

Verwenden Sie den WebSphere MQ-Ressourcenadapter nicht in WebSphere Application Server V6. Um in einer JMS-Anwendung von WebSphere Application Server aus auf die Ressourcen eines WebSphere MQ-Warteschlangenmanagers zuzugreifen, müssen Sie den WebSphere MQ-Messaging-Provider verwenden. Der WebSphere MQ-Messaging-Provider enthält eine Version von WebSphere MQ Classes for JMS.

WebSphere Application Server V7 enthält eine Version des WebSphere MQ-Ressourcenadapters der Version 7.

Weitere Informationen finden Sie in dem technischen Hinweis Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?.

WebSphere Application Server Liberty und der IBM WebSphere MQ-Ressourcenadapter
Der Ressourcenadapter von IBM WebSphere MQ Version 7.5 kann unter Verwendung der Komponente wmqJmsClient-1.1 in WebSphere Application Server Liberty Version 8.5.5, Fixpack 2 oder höher installiert werden. Sie können ihn jedoch auch (mit einigen Einschränkungen) mithilfe der generischen Java EE JCA-Unterstützung (Java Platform, Enterprise Edition Connector Architecture) installieren.

Allgemeine Einschränkungen bei der Installation des Ressourcenadapters in Liberty

Bei Verwendung der Komponente wmqJmsClient-1.1 und auch bei Verwendung der generischen JCA-Unterstützung gelten für den Ressourcenadapter von Version 7.5 die folgenden Einschränkungen:

- Die IBM WebSphere MQ classes for Java werden in Liberty nicht unterstützt. Sie dürfen weder zusammen mit der Messaging-Komponente 'IBM WebSphere 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 WebSphere MQ-Ressourcenadapter hat den Transporttyp BINDINGS_THEN_CLIENT. Dieser Transporttyp wird von der Messaging-Komponente 'IBM WebSphere MQ Liberty' nicht unterstützt.

- Die Komponente 'IBM WebSphere MQ Advanced Message Security' (IBM WebSphere MQ AMS) ist nicht in der Messaging-Komponente 'IBM WebSphere MQ Liberty' enthalten.

Der Ressourcenadapter von IBM WebSphere MQ Version 7.5 kann nicht zusammen mit der Komponente wmqJmsClient-2.0 verwendet werden.

Konfiguration des WebSphere MQ-Ressourcenadapters

Zur Konfiguration des WebSphere MQ-Ressourcenadapters definieren Sie verschiedene JCA-Ressourcen und Systemeigenschaften.

Definieren Sie JCA-Ressourcen in den folgenden Kategorien:

- Eigenschaften des ResourceAdapter-Objekts, die die globalen Eigenschaften des Ressourcenadapters, z. B. die Diagnosetracestufe, darstellen. Eine Beschreibung dieser Eigenschaften finden Sie im Abschnitt „Konfiguration des ResourceAdapter-Objekts" auf Seite 781.
- Eigenschaften eines ActivationSpec-Objekts, die festlegen, wie eine MDB für die eingehende Kommunikation aktiviert wird. Eine Beschreibung dieser Eigenschaften finden Sie im Abschnitt „Ressourcenadapter für eingehende Kommunikation konfigurieren" auf Seite 782.
- Eigenschaften eines ConnectionFactory-Objekts, die vom Anwendungsserver zur Erstellung eines JMS-ConnectionFactory-Objekts für die abgehende Kommunikation verwendet werden. Eine Beschreibung dieser Eigenschaften finden Sie im Abschnitt „Ressourcenadapter für abgehende Kommunikation konfigurieren" auf Seite 798.
- Eigenschaften eines verwalteten Destination-Objekts, die vom Anwendungsserver zur Erstellung eines JMS-Queue-Objekts oder JMS-Topic-Objekts für die abgehende Kommunikation verwendet werden. Eine Beschreibung dieser Eigenschaften finden Sie auch unter „Ressourcenadapter für abgehende Kommunikation konfigurieren" auf Seite 798.

Die RAR-Datei des WebSphere 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://java.sun.com/xml/ns/j2ee/connector_1_5.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. Zum Beispiel erfordert WebSphere Application Server Community Edition den Implementierungsplan geronimo-ra.xml.

Wenn Sie Secure Sockets Layer (SSL) 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 kann sich daher auf den Anwendungsserver auswirken, wenn andere Anwendungen, die auf dem Anwendungsserver ausgeführt werden, SSL-Verbindungen verwenden. Der Anwendungsserver könnte diese Eigenschaften auch auf andere Werte zurücksetzen. Weitere Informationen zur Verwendung von SSL mit WebSphere MQ Classes for JMS finden Sie im Abschnitt „Secure Sockets Layer (SSL) mit WebSphere MQ Classes for JMS verwenden" auf Seite 963.

Mit dem WebSphere MQ-Ressourcenadapter wird ein Installationsprüftestprogramm (Installation Verification Test Program, IVT-Programm) bereitgestellt. Aber bevor Sie das Programm ausführen können, müssen Sie den Ressourcenadapter konfigurieren. Sie finden Informationen darüber, was Sie für die Ausführung des IVT-Programms konfigurieren müssen, im Abschnitt „Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter" auf Seite 832.

Die Protokolle, Warnungen und Fehlernachrichten des Ressourcenadapters verwenden denselben Mechanismus wie IBM WebSphere MQ Classes for JMS. Ausführliche Informationen hierzu finden Sie im Abschnitt „Protokollierung und IBM WebSphere MQ classes for JMS" auf Seite 847. Bei WebSphere Ap-

plication Serverwerden diese Nachrichten automatisch an das Ausgabeprotokoll des Anwendungsservers umgeleitet. Für andere Anwendungsserver, wie z. B. WAS CE und JBoss, gehen diese standardmäßig in eine Datei mit dem Namen mqjms.log. Wenn Sie in der Konfiguration des Ressourcenadapters festlegen möchten, dass zusätzlich Warnungen im Standardausgabeprotokoll Ihres Anwendungsservers protokolliert werden sollen, legen Sie die folgende JVM-Systemeigenschaft für Ihren Anwendungsserver fest:

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

In der Dokumentation Ihres Anwendungsservers finden Sie Informationen zur Festlegung einer JVM-Systemeigenschaft.

Konfiguration des ResourceAdapter-Objekts
Das Objekt ResourceAdapter kapselt die globalen Eigenschaften des WebSphere MQ-Ressourcenadapters. Definieren Sie diese Eigenschaften mithilfe der Funktionen Ihres Ressourcenadapters.

Das ResourceAdapter-Objekt verfügt über zwei Eigenschaftsgruppen:

- Eigenschaften, die dem Diagnosetracing zugeordnet sind
- Eigenschaften, die dem Verbindungspool zugeordnet sind, der vom Ressourcenadapter verwaltet wird

Auf welche Weise Sie diese Eigenschaften definieren, hängt von der Verwaltungsschnittstelle ab, die von Ihrem Anwendungsserver bereitgestellt wird.

Weitere Informationen zur Definition von Eigenschaften für den Diagnosetrace finden Sie unter Traceerstellung für den IBM WebSphere MQ-Ressourcenadapter.

Der Ressourcenadapter verwaltet einen internen Verbindungspool von JMS-Verbindungen, die zur Übermittlung von Nachrichten an MDBs (Message-driven Beans) verwendet werden.Tabelle 95 auf Seite 781 In sind die Eigenschaften des ResourceAdapter -Objekts aufgelistet, die dem Verbindungspool zugeordnet sind.

Tabelle 95. Eigenschaften des ResourceAdapter-Objekts, die dem Verbindungspool zugeordnet sind

Name der Eigenschaft	**Typ**	**Standardwert**	**Beschreibung**
maxConnections	Zeichenfolge	50	Maximale Anzahl Verbindungen mit einem WebSphere MQ-Warteschlangenmanager und maximale Anzahl implementierter MDBs.
connectionConcurrency	Zeichenfolge	1	Maximale Anzahl MDBs für gemeinsame Nutzung einer JMS-Verbindung. Eine gemeinsame Nutzung von Verbindungen ist nicht möglich, sodass diese Eigenschaft immer den Wert 1 hat.
reconnectionRetryCount	Zeichenfolge	5	Maximale Anzahl Versuche, die der Ressourcenadapter unternimmt, um die Verbindung mit einem WebSphere MQ-Warteschlangenmanager nach einer Unterbrechung wiederherzustellen.
reconnectionRetryInterval	Zeichenfolge	300 000	Zeit in Millisekunden, die der Ressourcenadapter wartet, bevor er versucht, die Verbindung mit einem WebSphere 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.

Tabelle 95. Eigenschaften des ResourceAdapter-Objekts, die dem Verbindungspool zugeordnet sind (Forts.)			
Name der Eigenschaft	**Typ**	**Standardwert**	**Beschreibung**
startupRetryInterval	Zeichenfolge	30 000	Die standardmäßige Ruhezeit zwischen Verbindungsversuchen beim Startvorgang (in Millisekunden).

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 für den Fall, dass Verbindungen mit einem WebSphere MQ-Wartenschlangenmanager fehlschlagen, z. B. aufgrund eines Netzausfalls. 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. Um jedoch das Diagnosetracing auf UNIX and Linux-Systemen zu aktivieren, können Sie z. B. die folgenden Eigenschaften definieren:

```
traceEnabled:       true
traceLevel:         10
```

Diese Eigenschaften haben keine Wirkung, wenn der Ressourcenadapter nicht gestartet wurde, was beispielsweise dann der Fall ist, wenn Anwendungen, die WebSphere MQ-Ressourcen nutzen, nur im Client-Container ausgeführt werden. In dieser Situation können Sie die Eigenschaften für das Diagnosetracing als JVM-Systemeigenschaften (Java Virtual Machine) 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.

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 WebSphere 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 WebSphere 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.

In Tabelle 96 auf Seite 783 werden die Eigenschaften eines ActivationSpec-Objekts aufgelistet, die zum Erstellen einer JMS-Verbindung mit einem WebSphere MQ-Warteschlangenmanager verwendet werden.

Tabelle 96. 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 durch den Befehl **DISPLAY CONN MQSC/PCF** (mit dem Feld **APPLTAG**) oder in der Anzeige **Anwendungsverbindungen** von IBM WebSphere MQ Explorer (mit dem Feld **App name**) angezeigt.
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 96. 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 WebSphere MQ-Publish/Subscribe-Brokers. Oder für die Verwendung eines Brokers von WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker oder WebSphere Business Integration Message Broker im Kompatibilitätsmodus. Dieser Wert ist der Standardwert, falls TRANSPORT auf BIND oder CLIENT gesetzt ist. • **V2** - Für die Verwendung eines Brokers von WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker oder WebSphere Business Integration Message Broker 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** • Eine von der Java Virtual Machine (JVM) unterstützte ID des codierten Zeichensatzes	Die ID des codierten Zeichensatzes für eine Verbindung
Kanal	Zeichenfolge	• **SYSTEM.DEF.SVRCONN** • Der Name eines MQI-Kanals	Der Name des zu verwendenden MQI-Kanals
cleanupInterval [1]	int	• **3 600 000** • Eine positive Ganzzahl	Das in Millisekunden angegebene Intervall zwischen den Hintergrundausführungen des Publish/Subscribe-Bereinigungsdienstprogramms

Tabelle 96. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung einer JMS-Verbindung verwendet werden (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
cleanupLevel [1]	Zeichenfolge	• **SAFE** • KEINE • STRONG • FORCE • NONDUR	Die Bereinigungsstufe für einen brokerbasierten Subskriptionsspeicher
clientID	Zeichenfolge	• **Null** • Eine Client-ID	Die Client-ID für eine Verbindung
cloneSupport	Zeichenfolge	• **DISABLED** - Nur eine Instanz eines permanenten Topic-Subskribenten kann jeweils ausgeführt werden. • ENABLED-Mindestens zwei Instanzen desselben permanenten Topic-Subskribenten können gleichzeitig ausgeführt werden, aber jede Instanz muss in einer separaten JVM (Java Virtual Machine) 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 eingehende Kommunikation verwendet werden. Wenn 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.

Tabelle 96. 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
failIfQuiesce	Boolesch	• **wahr** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird
headerCompression	Zeichenfolge	• **Ohne** • SYSTEM - Es wird eine RLE-Komprimierung des Nachrichtenheaders durchgeführt.	Eine Liste der Verfahren, die zum Komprimieren von Headerdaten in einer Verbindung verwendet werden können
hostName	Zeichenfolge	• **localhost** • Ein Hostname • Eine IP-Adresse	Der Hostname oder die IP-Adresse des Systems, auf dem sich der Warteschlangenmanager befindet. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** ersetzt, wenn sie angegeben wird.
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.

Tabelle 96. 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
messageCompression	Zeichenfolge	• **Ohne** • 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	Bestimmt, ob die Nachrichtenauswahl durch WebSphere 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 96. 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
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** ersetzt, wenn sie angegeben wird.
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

Tabelle 96. 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
receiveExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren durch Kommas getrennten Elementen besteht, wobei jedes Element der vollständig qualifizierte Name einer Klasse ist, die die Schnittstelle MQReceiveExit der WebSphere MQ -Klassen für Java 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 96. 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 für die Auswahl der zu empfangenden Nachrichten verwendet, durchsucht WebSphere MQ Classes for JMS die WebSphere MQ-Warteschlange nach geeigneten Nachrichten in der Reihenfolge, die mit dem Attribut *MsgDeliverySequence* der Warteschlange festgelegt wurde. Wenn WebSphere MQ Classes for JMS eine geeignete Nachricht findet und diese an den Konsumenten übermittelt, setzt WebSphere MQ Classes for JMS die Suche nach der nächsten geeigneten Nachricht ab der aktuellen Position in der Warteschlange fort. WebSphere MQ Classes for JMS durchsucht die Warteschlange weiterhin auf diese Weise, bis das Ende der Warteschlange erreicht oder das durch den Wert dieser Eigenschaft festgelegte Zeitintervall in Millisekunden abgelaufen ist. In jedem Fall kehrt WebSphere 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 MQSecurityExit der WebSphere MQ -Klassen für Java 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 96. 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 durch Kommas getrennten Elementen besteht, wobei jedes Element der vollständig qualifizierte Name einer Klasse ist, die die Schnittstelle MQSendExit der WebSphere MQ -Klassen für Java implementiert.	Gibt ein Sendeexitprogramm oder eine Folge von Sendeexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen
sendExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Sendeexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden
shareConvAllowed	Boolesch	• NO-Eine Clientverbindung kann ihren Socket nicht gemeinsam nutzen. • **YES** -Eine Clientverbindung kann ihren Socket gemeinsam nutzen.	Gibt an, ob eine Clientverbindung ihr Socket gemeinsam mit anderen JMS-Verbindungen der obersten Ebene von demselben Prozess zu 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 (Lightweight Directory Access Protocol), die Zertifikatswiderruflisten für eine SSL-Verbindung enthalten.
sslCipherSuite	Zeichenfolge	• **Null** • Der Name einer Cipher-Suite	Die für eine SSL-Verbindung zu verwendende Cipher-Suite

Tabelle 96. 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 SSL-Verbindung eine CipherSuite verwenden muss, die vom IBM Java-JSSE-FIPS-Provider (IBMJSSEFIPS) unterstützt wird.
sslPeerName	Zeichenfolge	• **Null** • Eine Vorlage für definierte Namen	Eine Vorlage bei einer SSL-Verbindung, die verwendet wird, um den definierten Namen in dem vom Warteschlangenmanager bereitgestellten digitalen Zertifikat zu prüfen
sslResetCount	int	• **0** • Eine Ganzzahl im Bereich 0 bis 999 999 999	Die Gesamtzahl der von einer SSL-Verbindung gesendeten und empfangenen Bytes, bevor die von SSL verwendeten geheimen Schlüssel erneut vereinbart werden
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 WebSphere MQ Classes for JMS persistente Daten über aktive Subskriptionen speichert

Tabelle 96. 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. Wenn der Wert BINDINGS_THEN_CLIENT angegeben wird, versucht der Ressourcenadapter zuerst, eine Verbindung in Bindungsmodus herzustellen. Schlägt dieser Verbindungsversuch fehlt, versucht der Ressourcenadapter, eine Verbindung im Clientmodus herzustellen.
username	Zeichenfolge	• **Null** • Ein Benutzername	Der beim Herstellen einer Verbindung zu einem Warteschlangenmanager zu verwendende Standardbenutzername
wildcardFormat	Zeichenfolge	• CHAR- Erkennt nur Zeichenplatzhalter (wie Broker Version 1) • **TOPIC** - Erkennt nur Platzhalter auf Themenebene (wie in Broker Version 2)	Gibt die zu verwendende Version der Platzhaltersyntax an

Anmerkungen:

1. Diese Eigenschaft kann mit Version 7.0 von WebSphere MQ Classes for JMS verwendet werden. Sie hat keine Auswirkung auf eine Anwendung, die mit einem Warteschlangenmanager der Version 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 bei Verwendung des IBM WebSphere MQ-Ressourcenadapters” auf Seite 820.
3. Informationen zur Konfiguration des Ressourcenadapters zur Lokalisierung eines Exits finden Sie im Abschnitt „IBM WebSphere MQ classes for JMS für Verwendung von Kanalexits konfigurieren” auf Seite 970.

In Tabelle 97 auf Seite 793 werden die Eigenschaften eines ActivationSpec-Objekts aufgelistet, die zum Erstellen eines JMS-Verbindungskonsumenten verwendet werden.

Tabelle 97. Eigenschaften eines ActivationSpec-Objekts, die für die Erstellung eines JMS-Verbindungskonsumenten verwendet werden

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
destination	Zeichenfolge	Der Name des Ziels	Das Ziel, von dem Nachrichten empfangen werden sollen. Die Eigenschaft 'useJNDI' legt fest, wie der Wert dieser Eigenschaft interpretiert wird.

Tabelle 97. 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
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.
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 nur gültig, wenn die Anwendung in WebSphere Application Server Version 7 oder höher ausgeführt wird.
nonASFRollbackEnabled	Boolesch	• **false** (false) - Die Nachricht wird selbst dann verarbeitet, wenn die MDB fehlschlägt • true (wahr) - Bei einem Fehler innerhalb der MDB wird die Nachricht mit einem Rollback in die Warteschlange zurückgesetzt.	Gibt an, ob die Nachrichtenzustellung innerhalb eines WebSphere 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 97. 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, es sei denn, **`receiveConversion`** wurde auf `QMGR` gesetzt.
receiveConversion	Zeichenfolge	• **CLIENT_MSG** • QMGR	Eine Zieleigenschaft, die bestimmt, ob eine Datenkonvertierung vom Warteschlangenmanager durchgeführt wird.
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.

Tabelle 97. 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
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
useJNDI	Boolesch	• **false** (falsch) - Die Eigenschaft 'destination' wird als Name einer Warteschlange oder eines Topics von WebSphere 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

Die ActivationSpec-Eigenschaften mit den Namen 'destination' und 'destinationType' müssen explizit definiert werden. Alle anderen Eigenschaften sind optional.

Ein ActivationSpec-Objekt kann Eigenschaften aufweisen, die miteinander in Konflikt stehen. Sie können beispielsweise SSL-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, was angibt, dass Sie permanente Subskriptionen verwenden möchten, müssen Sie auch die Eigenschaft 'subscriptionName' definieren.

Wenn die Eigenschaften 'ccdtURL' und 'channel' beide definiert sind, wird die Ausnahmebedingung 'InvalidPropertyException' ausgelöst. Wenn Sie jedoch nur die Eigenschaft 'ccdtURL' definieren und für die Eigenschaft mit dem Namen 'channel' den Standardwert SYSTEM.DEF.SVRCONN übernehmen, wird keine

Ausnahmebedingung ausgelöst und die über die Eigenschaft 'ccdtURL' angegebene Definitionstabelle für Clientkanäle wird zum Starten einer JMS-Verbindung verwendet.

Die meisten Eigenschaften eines ActivationSpec-Objekts entsprechen den Eigenschaften der WebSphere MQ Classes for JMS-Objekte oder Parameter von WebSphere MQ Classes for JMS-Methoden. Es gibt jedoch drei Optimierungseigenschaften und eine Nutzbarkeitseigenschaft, die keine funktionalen Entsprechungen in WebSphere 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
Eine Beschreibung dieser Eigenschaft finden Sie im Abschnitt Tabelle 97 auf Seite 793.

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 Nachrich-

ten 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 „Installation des WebSphere MQ-Ressourcenadapters” auf Seite 778.

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.

Behandlung nicht verarbeitbarer eingehender Nachrichten im Ressourcenadapter

Gelegentlich kann es vorkommen, dass eine Nachricht, die an eine MDB zugestellt wurde, mit einem Rollback in eine WebSphere 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 WebSphere MQ festlegen, dass WebSphere MQ Classes for JMS eine nicht verarbeitbare Nachricht automatisch an eine andere Nachricht zur Untersuchung überträgt oder dass die Nachricht gelöscht wird.

Sie finden Details zur Behandlung nicht verarbeitbarer Nachrichten im Abschnitt „Handhabung nicht verarbeitbarer Nachrichten in IBM WebSphere MQ classes for JMS” auf Seite 945.

Zugehörige Tasks

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für UNIX, Linux und Windows

Ressourcenadapter für abgehende Kommunikation konfigurieren

Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

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 ConnectionFactory-Objekt und ein Warteschlangenobjekt (Queue-Objekt) aus dem JNDI-Namensbereich ab und verwendet die Objekte für das Senden einer Nachricht an eine WebSphere MQ-Warteschlange. Das Servlet empfängt dann die gesendete Nachricht.

Um die abgehende Kommunikation zu konfigurieren, definieren Sie JCA-Ressourcen in den folgenden Kategorien:

- Eigenschaften eines ConnectionFactory-Objekts, die vom Anwendungsserver für die Erstellung eines JMS-ConnectionFactory-Objekts verwendet werden.
- Eigenschaften eines verwalteten Destination-Objekts, die vom Anwendungsserver für die Erstellung eines JMS-Queue-Objekts oder JMS-Topic-Objekts verwendet werden.

Auf welche Weise Sie diese Eigenschaften definieren, hängt von der Verwaltungsschnittstelle ab, die von Ihrem Anwendungsserver bereitgestellt wird. ConnectionFactory-, Queue- und Topic-Objekte, die vom Anwendungsserver erstellt wurden, werden an einen JNDI-Namensbereich gebunden, aus dem sie von einer Anwendung abgerufen werden können.

In der Regel definieren Sie ein ConnectionFactory-Objekt für jeden Warteschlangenmanager, zu dem Anwendungen möglicherweise eine Verbindung herstellen. Sie definieren ein Queue-Objekt für jede Warteschlange, auf die Anwendungen in der Punkt-zu-Punkt-Domäne möglicherweise zugreifen möchten. Zudem definieren Sie ein Topic-Objekt für jedes Thema, das Anwendungen möglicherweise veröffentlichen oder subskribieren möchten. Ein ConnectionFactory-Objekt kann domänenunabhängig sein. Alternativ kann es auch domänenspezifisch sein, also ein QueueConnectionFactory-Objekt für die Punkt-zu-Punkt-Domäne oder ein TopicConnectionFactory-Objekt für die Publish/Subscribe-Domäne.

In Tabelle 98 auf Seite 799 werden die Eigenschaften eines ConnectionFactory-Objekts aufgelistet.

Tabelle 98. 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`** (mit dem Feld **`APPLTAG`**) oder in der Anzeige **Anwendungsverbindungen** von IBM WebSphere MQ Explorer (mit dem Feld **App name**) angezeigt.
brokerCCSubQueue [1]	Zeichenfolge	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Ein Warteschlangenname	Der Name der Warteschlange, aus der ein Verbindungskonsument nicht permanente Subskriptionsnachrichten empfängt.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
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 WebSphere MQ-Publish/Subscribe-Brokers. Oder für die Verwendung eines Brokers von IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker oder WebSphere Business Integration Message Broker im Kompatibilitätsmodus. Dieser Wert ist der Standardwert, falls TRANSPORT auf BIND oder CLIENT gesetzt ist. • **V2** - Für die Verwendung eines Brokers von IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker oder WebSphere Business Integration Message Broker im nativen Modus. Dieser Wert ist der Standardwert, falls TRANSPORT auf DIRECT oder DIRECTHTTP gesetzt ist.	Die Version des verwendeten Brokers.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
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.
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.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (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 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	• **wahr** • false	Gibt an, ob Aufrufe bestimmter Methoden fehlschlagen, wenn der Warteschlangenmanager gerade stillgelegt wird.
headerCompression	Zeichenfolge	• **Ohne** • SYSTEM - Es wird eine RLE-Komprimierung des Nachrichtenheaders durchgeführt.	Eine Liste der Verfahren, die zum Komprimieren von Headerdaten in einer Verbindung verwendet werden können.
hostName	Zeichenfolge	• **localhost** • Ein Hostname • Eine IP-Adresse	Der Hostname oder die IP-Adresse des Systems, auf dem sich der Warteschlangenmanager befindet. Die Eigenschaften **hostname** und **port** werden durch die Eigenschaft **connectionNameList** ersetzt, wenn sie angegeben wird.

Tabelle 98. 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	• **Ohne** • 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	Bestimmt, ob die Nachrichtenauswahl durch IBM WebSphere 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 98. 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** ersetzt, wenn sie angegeben wird.
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 WebSphere 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 98. 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 WebSphere 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, durchsucht WebSphere MQ Classes for JMS die IBM WebSphere MQ-Warteschlange nach geeigneten Nachrichten in der Reihenfolge, die mit dem Attribut *MsgDeliverySequence* der Warteschlange festgelegt wurde. Wenn WebSphere MQ Classes for JMS eine geeignete Nachricht findet und diese an den Konsumenten übermittelt, setzt WebSphere MQ Classes for JMS die Suche nach der nächsten geeigneten Nachricht ab der aktuellen Position in der Warteschlange fort. WebSphere MQ Classes for JMS durchsucht die Warteschlange weiterhin auf diese Weise, bis das Ende der Warteschlange erreicht oder das durch den Wert dieser Eigenschaft festgelegte Zeitintervall in Millisekunden abgelaufen ist. In jedem Fall kehrt WebSphere MQ Classes for JMS zum Anfang der Warteschlange zurück, um die Suche fortzusetzen, und es beginnt ein neues Zeitintervall.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
securityExit [3]	Zeichenfolge	• **Null** • Der vollständig qualifizierte Name einer Klasse, die die Schnittstelle MQSecurityExit der WebSphere MQ -Klassen für Java implementiert.	Gibt ein Sicherheitsexitprogramm für den Kanal an.
securityExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge mit Benutzerdaten	Die Benutzerdaten, die an ein Sicherheitsexitprogramm des Kanals übergeben werden, wenn dieses aufgerufen wird.
sendCheckCount	int	• **0** • Jede beliebige positive Ganzzahl.	Die Anzahl der Sendeaufrufe, die zwischen den Prüfungen auf Fehler innerhalb einer einzelnen JMS-Sitzung ohne Transaktion bei der asynchronen PUT-Operation zugelassen werden sollen.
sendExit [3]	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren durch Kommas getrennten Elementen besteht, wobei jedes Element der vollständig qualifizierte Name einer Klasse ist, die die Schnittstelle MQSendExit der WebSphere MQ -Klassen für Java implementiert.	Gibt ein Sendeexitprogramm oder eine Folge von Sendeexitprogrammen für den Kanal an, die nacheinander ausgeführt werden sollen.
sendExitInit	Zeichenfolge	• **Null** • Eine Zeichenfolge, die aus einem oder mehreren Elementen von Benutzerdaten besteht, die durch Kommas voneinander getrennt sind	Die Benutzerdaten, die an die Sendeexitprogramme des Kanals gesendet werden, wenn diese aufgerufen werden.
shareConvAllowed	Boolesch	• NO-Eine Clientverbindung kann ihren Socket nicht gemeinsam nutzen. • **YES** -Eine Clientverbindung kann ihren Socket gemeinsam nutzen.	Gibt an, ob eine Clientverbindung ihr Socket gemeinsam mit anderen JMS-Verbindungen der obersten Ebene von demselben Prozess zu 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.	Diese Eigenschaft steuert die Richtlinie für den Nachrichtenabruf eines TopicSubscriber-Objekts.

Tabelle 98. 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 (Lightweight Directory Access Protocol), die Zertifikatswiderruflisten für eine SSL-Verbindung enthalten.
sslCipherSuite	Zeichenfolge	• **Null** • Der Name einer Cipher-Suite	Die Cipher-Suite, die für eine SSL-Verbindung verwendet werden soll.
sslFipsRequired [2]	Boolesch	• **false** • true	Gibt an, ob eine SSL-Verbindung eine CipherSuite verwenden muss, die vom IBM Java -JSSE-FIPS-Provider (IBMJSSEFIPS) unterstützt wird.
sslPeerName	Zeichenfolge	• **Null** • Eine Vorlage für definierte Namen	Eine Vorlage bei einer SSL-Verbindung, die verwendet wird, um den definierten Namen in dem vom Warteschlangenmanager bereitgestellten digitalen Zertifikat zu prüfen.
sslResetCount	int	• **0** • Eine Ganzzahl im Bereich 0 bis 999 999 999	Die Gesamtzahl der von einer SSL-Verbindung gesendeten und empfangenen Bytes, bevor die von SSL verwendeten geheimen Schlüssel erneut vereinbart werden.
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.
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.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
subscriptionStore [1]	Zeichenfolge	• **Broker** • MIGRATE • WARTESCHLANGE	Legt fest, ob WebSphere MQ Classes for JMS persistente Daten über aktive Subskriptionen speichert.
targetClientMatching	Boolesch	• **wahr** • false	Gibt an, ob eine Antwortnachricht, die an die im Headerfeld 'JMSReplyTo' einer eingehenden Nachricht angegebene Warteschlange gesendet wird, nur dann einen MQRFH2-Header enthält, wenn die eingehende Nachricht einen MQRFH2-Header enthält.

Tabelle 98. 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, bei der 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. Wenn die neue Modellwarte-

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)

Name der Eigenschaft	Typ	Gültige Werte (der Standardwert ist fett dargestellt)	Beschreibung
tempQPrefix	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Präfix, das verwendet werden kann, um den Namen einer dynamischen IBM WebSphere MQ -Warteschlange zu bilden Die Regeln für die Bildung des Präfix entsprechen den Regeln für die Bildung des Inhalts des Felds *DynamicQName* in einem IBM WebSphere MQ -Objektdeskriptor (Struktur MQOD), aber das letzte nicht leere Zeichen muss ein Stern (*) sein. Wenn der Wert der Eigenschaft eine leere Zeichenfolge ist, verwendet WebSphere MQ Classes for JMS den Wert AMQ.* beim Erstellen einer dynamischen Warteschlange.	Das Präfix, das verwendet wird, um den Namen einer dynamischen IBM WebSphere MQ-Warteschlange zu bilden.
tempTopicPrefix	Zeichenfolge	Eine beliebige Zeichenfolge ungleich null, die nur aus gültigen Zeichen für eine IBM WebSphere 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 Wertes für die Eigenschaft TEMPTOPICPREFIX können Sie spezifische Modellwarteschlangen zum Erstellen der verwalteten Warteschlangen für Subskribenten von temporären Themen unter Verwendung der jeweiligen Verbindung definieren.
transportType	Zeichenfolge	• **Client** • BINDINGS • BINDINGS_THEN_CLIENT	Gibt an, ob eine Verbindung mit einem Warteschlangenmanager den Clientmodus oder den Bindungsmodus verwendet. Wenn der Wert BINDINGS_THEN_CLIENT angegeben wird, versucht der Ressourcenadapter zuerst, eine Verbindung in Bindungsmodus herzustellen. Schlägt dieser Verbindungsversuch fehl, versucht er, eine Verbindung im Clientmodus herzustellen.
username	Zeichenfolge	• **Null** • Ein Benutzername	Der beim Herstellen einer Verbindung zu einem Warteschlangenmanager zu verwendende Standardbenutzername.

Tabelle 98. Eigenschaften eines ConnectionFactory-Objekts (Forts.)			
Name der Eigenschaft	**Typ**	**Gültige Werte (der Standardwert ist fett dargestellt)**	**Beschreibung**
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 bei Version 7 von IBM WebSphere MQ classes for JMS verwendet werden, sie hat jedoch keine Auswirkung auf eine Anwendung, die mit einem Warteschlangenmanager der Version 7 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 bei Verwendung des IBM WebSphere MQ-Ressourcenadapters” auf Seite 820.
3. Informationen zur Konfiguration des Ressourcenadapters zur Lokalisierung eines Exits finden Sie im Abschnitt „IBM WebSphere MQ classes for JMS für Verwendung von Kanalexits konfigurieren” auf Seite 970.

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
```

In Tabelle 99 auf Seite 811 werden die Eigenschaften aufgelistet, die sowohl für ein Queue-Objekt als auch für ein Topic-Objekt verwendet werden.

Tabelle 99. 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.

Tabelle 99. 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
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 eine zSeries-Codierung. NATIVE ist äquivalent zu der Zeichenfolge NNN.	Die Darstellung von binären Ganzzahlen, Ganzzahlen im gepackten Dezimalformat und Gleitkommazahlen für das Ziel.
expiry	Zeichenfolge	• **APP** - Die Ablaufzeit einer Nachricht wird durch den Nachrichtenproduzenten bestimmt. • UNLIM - Eine Nachricht läuft nie ab. • 0 - Eine Nachricht läuft nie ab. • Eine positive Ganzzahl, die die Ablaufzeit einer Nachricht in Millisekunden darstellt.	Die Ablaufzeit einer Nachricht, die an das Ziel gesendet wird.
failIfQuiesce	Zeichenfolge	• **wahr** • false	Gibt an, ob ein Zugriffsversuch auf das Ziel fehlschlägt, wenn sich der Warteschlangenmanager im Stilllegungsmodus befindet.

Tabelle 99. 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 WebSphere 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 WebSphere MQ-Warteschlange entsprechend der Erläuterung in „Persistente JMS-Nachrichten” auf Seite 961 bestimmt.	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 WebSphere 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 99. 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, es sei denn, **`receiveConversion`** wurde auf `QMGR` gesetzt.
receiveConversion	Zeichenfolge	• **CLIENT_MSG** • QMGR	Eine Zieleigenschaft, die bestimmt, ob eine Datenkonvertierung vom Warteschlangenmanager durchgeführt wird.
targetClient	Zeichenfolge	• **JMS** - Das Ziel einer Nachricht ist eine JMS-Anwendung. • MQ -Das Ziel einer Nachricht ist eine Nicht-JMS- IBM WebSphere MQ -Anwendung.	Gibt an, ob das Ziel einer Nachricht, die an das Ziel gesendet wird, eine JMS-Anwendung ist. Eine Nachricht, deren Ziel eine JMS-Anwendung ist, enthält einen MQRFH2-Header.

In Tabelle 100 auf Seite 814 werden die Eigenschaften aufgelistet, die sich speziell auf ein Queue-Objekt beziehen.

Tabelle 100. 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 WebSphere MQ-Warteschlange ist.
baseQueueName	Zeichenfolge	• **"" (leere Zeichenfolge)** • Ein Warteschlangenname	Der Name der zugrunde liegenden IBM WebSphere MQ -Warteschlange.

In Tabelle 101 auf Seite 815 werden die Eigenschaften aufgelistet, die sich speziell auf ein Topic-Objekt beziehen.

Tabelle 101. 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 in der Beschreibung der Eigenschaft BROKEDURRSUBQ in der Dokumentation zu WebSphere MQ Explorer.
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. Wenn Sie jedoch den Wert dieser Eigenschaft nicht festlegen, 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 101. 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. Wenn Sie jedoch den Wert dieser Eigenschaft nicht festlegen, wird stattdessen der Wert der Eigenschaft 'brokerVersion' des ConnectionFactory-Objekts verwendet.

Anmerkung:

1. Diese Eigenschaft kann bei Version 7 von IBM WebSphere MQ classes for JMS verwendet werden, sie hat jedoch keine Auswirkung auf eine Anwendung, die mit einem Warteschlangenmanager der Version 7 verbunden ist, es sei denn, die Eigenschaft 'providerVersion' des ConnectionFactory-Objekts wurde auf eine niedrigere Versionsnummer 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 Tasks

Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden

Zugehörige Verweise

Federal Information Processing Standards (FIPS) für UNIX, Linux und Windows

V 7.5.0.9 *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 WebSphere 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 für WebSphere Application Server Traditional und für WebSphere Application Server Liberty 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 WebSphere 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

- In WebSphere Application Server Traditional wird die Eigenschaft **targetClientMatching** über die Verwaltungskonsole als angepasste Eigenschaft in der IBM WebSphere MQ-Aktivierungsspezifikation definiert:
  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`.
- In WebSphere Application Server Liberty wird die Eigenschaft **targetClientMatching** in der Definition einer Aktivierungsspezifikation in der Datei `server.xml` definiert.

  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 933
Anstatt Ziele als verwaltete Objekte aus einem JNDI-Namensbereich abzurufen, kann eine JMS-Anwendung eine Sitzung verwenden, um Ziele dynamisch zur Laufzeit zu erstellen. Eine Anwendung kann einen Uniform Resource Identifier (URI) verwenden, um eine WebSphere MQ-Warteschlange oder ein Thema und optional eine oder mehrere Eigenschaften eines Queue- oder Topic-Objekts anzugeben.

„Ressourcenadapter für abgehende Kommunikation konfigurieren” auf Seite 798
Definieren Sie die Eigenschaften eines ConnectionFactory-Objekts und eines verwalteten Zielobjekts, um die abgehende Kommunikation zu konfigurieren.

ASF-Modus und Modus ohne ASF
Der ASF-Modus (ASF = Application Server Facilities) stellt das Standardverfahren für die Nachrichtenverarbeitung durch den Nachrichten-Listener-Service in WebSphere Application Server dar.

Es gibt zwei Betriebsarten für den Nachrichten-Listener-Service: Application Server Facilities (ASF) und ohne Application Server Facilities (ohne ASF):

- Der ASF-Modus bietet gemeinsamen Zugriff und Transaktionsunterstützung für Anwendungen. Für Publish/Subscribe-Message-driven Beans ermöglicht der ASF-Modus einen besseren Durchsatz und gemeinsamen Zugriff, da der Listener im Modus ohne ASF ein Singlethread-Listener ist.
- Der Modus ohne ASF ist hauptsächlich für die Verwendung mit Messaging-Providern anderer Anbieter vorgesehen, die JMS ASF, eine optionale Erweiterung zur JMS-Spezifikation, nicht unterstützen. Auch

der Modus ohne ASF ist transaktionsorientiert, da der Pfad jedoch kürzer als beim ASF-Modus ist, ist im Allgemeinen eine bessere Leistung möglich.

Legen Sie für diese Eigenschaft einen Wert ungleich null fest, um die Betriebsart ohne ASF für alle Message-driven Bean-Listener auf dem Anwendungsserver zu ermöglichen.

Anmerkung:

Auf z/OS-Systemen kann der Modus ohne ASF nicht ausgewählt werden, in diesem Fall darf also kein Wert ungleich null für diese Eigenschaft festgelegt werden.

Nachrichtenverarbeitung im ASF-Modus

Im ASF-Modus werden Serversitzungen und Threads nur für Arbeit zugeordnet, wenn eine für die Message-driven Bean (MDB) geeignete Nachricht erkannt wird. Die Anzahl der Threads, die eine MDB gleichzeitig verarbeiten kann, wird durch den Wert der Eigenschaft **Maximum Sessions** für den Listener-Port oder die Aktivierungsspezifikation bestimmt.

Nachrichtenverarbeitung im Modus ohne ASF

Im Modus ohne ASF sind die Threads ab dem Moment aktiv, da der Listener-Port oder die Aktivierungsspezifikation gestartet wird. Die Anzahl aktiver Threads wird durch den Wert der Eigenschaft **Maximum Sessions** (Maximale Sitzungsanzahl) festgelegt. Unabhängig von der Zahl der zur Verarbeitung verfügbaren Nachrichten ist die in der Eigenschaft **Maximum Sessions** angegebene Anzahl Threads aktiv. Bei jedem aktiven Thread handelt es sich um eine einzelne physische Netzverbindung.

In IBM WebSphere MQ Version 7.0 oder höher können bis zu zehn Threads eine einzelne physische Netzverbindung gemeinsam nutzen.

Zugehörige Konzepte

IBM WebSphere MQ-Klassen für JMS - Anwendungsserverfunktionen

In diesem Abschnitt wird beschrieben, wie WebSphere MQ-Klassen für JMS die ConnectionConsumer-Klasse und erweiterte Funktionalität in der Session-Klasse implementiert. Darüber hinaus enthält der Abschnitt eine Zusammenfassung der Funktion eines Serversitzungspools.

Zugehörige Tasks

Aktivierungsspezifikationen für den Modus ohne ASF konfigurieren

Aktivierungsspezifikationen bieten eine standardisierte Möglichkeit der Verwaltung und Konfiguration der Beziehung zwischen einer Message-driven Bean (MDB), die in WebSphere Application Server ausgeführt wird, und einem Ziel in IBM WebSphere MQ. In diesem Abschnitt wird die Konfiguration von WebSphere Application Server für die Verarbeitung von Nachrichten im Modus ohne ASF erläutert.

Zugehörige Informationen

Nachrichtenverarbeitung im ASF-Modus und im Modus ohne ASF

Aktivierungsspezifikationen für den Modus ohne ASF konfigurieren

Aktivierungsspezifikationen bieten eine standardisierte Möglichkeit der Verwaltung und Konfiguration der Beziehung zwischen einer Message-driven Bean (MDB), die in WebSphere Application Server ausgeführt wird, und einem Ziel in IBM WebSphere MQ. In diesem Abschnitt wird die Konfiguration von WebSphere Application Server für die Verarbeitung von Nachrichten im Modus ohne ASF erläutert.

Vorbereitende Schritte

Wie die Eigenschaften einer Aktivierungsspezifikation definiert werden, ist von den Verwaltungsschnittstellen abhängig, die Ihr Anwendungsserver bereitstellt. Bei dieser Task wird vorausgesetzt, dass Sie WebSphere Application Server Version 7 oder höher als Anwendungsserver und IBM WebSphere MQ als Messaging-Provider verwenden.

Anmerkung:

Auf z/OS-Systemen kann der Modus ohne ASF nicht ausgewählt werden.

Informationen zu diesem Vorgang

Über die Eigenschaften einer Aktivierungsspezifikation wird festgelegt, wie eine MDB (Message-driven Bean) JMS-Nachrichten aus einer IBM WebSphere MQ-Warteschlange empfängt. Definieren Sie die Eigenschaften einer oder mehrerer Aktivierungsspezifikationen, um den Modus ohne ASF zu konfigurieren.

Es gibt mehrere IBM WebSphere MQ-Konfigurationen, die im Modus ohne ASF verwendet werden können. Bei den folgenden Konfigurationen verwendet jeder Thread eine separate physische Netzverbindung:

- Einen IBM WebSphere MQ-Warteschlangenmanager der Version 7.x, der eine Verbindungsfactory verwendet, bei der die Eigenschaft für die Providerversion auf 6 gesetzt ist.
- Ein IBM WebSphere MQ-Warteschlangenmanager der Version 7.x, der eine Verbindungsfactory verwendet, bei der die Eigenschaft für die Providerversion auf 6 oder auf keine Version gesetzt ist, und der eine Verbindung über einen IBM WebSphere MQ-Kanal herstellt, für den der Parameter **SHARECNV** (gemeinsame Dialognutzung) auf 0 gesetzt ist.

Legen Sie für die Konfiguration des Modus ohne ASF für die Aktivierungsspezifikationseigenschaft **NON.ASF.RECEIVE.TIMEOUT** eine positive ganze Zahl fest, die angibt, dass die Bereitstellung ohne ASF 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. Weitere Informationen finden Sie unter **Message listener service custom properties**.

Dieser Parameter ist nur gültig, wenn die Anwendung auf WebSphere Application Server Version 7 oder höher ausgeführt wird.

Vorgehensweise

1. Starten Sie die Verwaltungskonsole von WebSphere Application Server.
2. Rufen Sie die Seite mit den Einstellungen für den Listener-Service auf:
   a) Wählen Sie im Navigationsfenster **Server > Servertypen > WebSphere -Anwendungsserver**aus.
   b) Klicken Sie im Inhaltsteilfenster auf den Namen des Anwendungsservers.
   c) Klicken Sie unter **Kommunikation**auf **Messaging > Message Listener Service**.
3. Legen Sie die angepasste Eigenschaft **NON.ASF.RECEIVE.TIMEOUT** als angepasste Eigenschaft des Nachrichten-Listener-Service fest.
   a) Klicken Sie auf **Custom properties** (Angepasste Eigenschaften).
   b) Klicken Sie auf **Neu**.
   c) Geben Sie den Namen der Eigenschaft, **NON.ASF.RECEIVE.TIMEOUT**, in das Feld **Name** ein.
   d) Geben Sie in das Feld **Value** (Wert) den erforderlichen Wert ein.
   e) Klicken Sie auf **OK**.
4. Speichern Sie Ihre Änderungen der Hauptkonfiguration.
5. Stoppen und starten Sie den Anwendungsserver erneut, um die geänderte Konfiguration zu aktivieren.

Ergebnisse

Sie haben die Eigenschaften des Nachrichten-Listener-Service für WebSphere Application Server für die Verwendung des Modus ohne ASF konfiguriert.

Anmerkung: Wenn Sie den Modus ohne ASF verwenden, müssen Sie sicherstellen, dass Sie genügend Zeit für die Verarbeitung einräumen, damit diese abgeschlossen werden kann, bevor die Gesamtlaufzeit der Transaktion erreicht wird, um unerwünschte Zeitlimitüberschreitungen bei Transaktionen zu vermeiden. Weitere Informationen finden Sie unter **NON.ASF.RECEIVE.TIMEOUT** in der Produktdokumentation für WebSphere Application Server.

Zugehörige Konzepte

„ASF-Modus und Modus ohne ASF” auf Seite 817

Der ASF-Modus (ASF = Application Server Facilities) stellt das Standardverfahren für die Nachrichtenverarbeitung durch den Nachrichten-Listener-Service in WebSphere Application Server dar.

Ressourcenadapter für eingehende Kommunikation konfigurieren
Definieren Sie die Eigenschaften eines oder mehrerer ActivationSpec-Objekte, um die eingehende Kommunikation zu konfigurieren.

Zugehörige Informationen
Message-driven Beans
Nachrichten-Listener-Service
Nachrichtenverarbeitung im ASF-Modus und im Modus ohne ASF
Verarbeitung von Nachrichten im Modus ohne ASF

Einschränkungen bei Verwendung des IBM WebSphere MQ-Ressourcenadapters

Wenn Sie den IBM WebSphere MQ-Ressourcenadapter verwenden, sind manche Funktionen von IBM WebSphere MQ nicht oder nur eingeschränkt verfügbar.

Bei dem IBM WebSphere MQ-Ressourcenadapter bestehen folgende Einschränkungen:

- Der Ressourcenadapter IBM WebSphere MQ wird auf allen IBM WebSphere MQ -Plattformen mit Ausnahme von z/OSunterstützt.
- Der IBM WebSphere MQ-Ressourcenadapter unterstützt keine Echtzeitverbindungen zu einem Broker. Er unterstützt nur Verbindungen zu einem IBM WebSphere MQ-Warteschlangenmanager im Client- oder Bindungsmodus.
- Der IBM WebSphere 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 auf 'true' bzw. für alle JCA-Ressourcen auf 'false' gesetzt 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 WebSphere MQ den Ursachencode MQRC_UNSUPPORTED_CIPHER_SUITE aus. Dies ist selbst dann der Fall, wenn eine SSL-Verbindung nicht 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. Für WebSphere Application Server gilt diese Einschränkung nicht.
- 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 Version 7.0.1 eingeführte Funktionalität für die Verbindungswiederholung wird für abgehende Verbindungen nicht unterstützt, wenn die Ausführung in einem JEE-Container (EJB/Servlet) erfolgt. Die Verbindungswiederholung wird für die abgehende JMS-Verarbeitung grundsätzlich nicht unterstützt, wenn der Adapter im Kontext eines JEE-Containers verwendet wird, und zwar unabhängig von der Transaktionskonfiguration oder einer nicht transaktionsorientierten Verwendung.

Zugehörige Tasks
Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden
Zugehörige Verweise
Federal Information Processing Standards (FIPS) für UNIX, Linux und Windows

Konfiguration von Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, im Anschluss an die Installation

In diesem Abschnitt erfahren Sie, welche Berechtigungen für Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, erforderlich sind, 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.

Denken Sie daran, die Readme-Datei zu WebSphere MQ zu lesen. 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.

Details zur Zugriffssteuerung in IBM WebSphere MQfinden Sie unter Sicherheit unter Windows einrichten, UNIX and Linux -Systeme .

WebSphere MQ Classes for JMS-Anwendungen benötigen die Berechtigungen `connect` und `inq` für den Warteschlangenmanager. Sie können die entsprechenden Berechtigungen mit dem Steuerbefehl **setmqaut** 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 die WebSphere MQ-Klassen für JMS eine Verbindung zu einem IBM WebSphere MQ-Warteschlangenmanager im Migrationsmodus des IBM WebSphere MQ-Messaging-Providers herstellen:

- 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 WebSphere MQ-Messaging-Providers finden Sie im Abschnitt Zeitpunkt für die Verwendung von **PROVIDERVERSION**.

Wenn WebSphere MQ Classes for JMS in diesem Modus eine Verbindung zu einem Warteschlangenmanager herstellt, 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 WebSphere MQ-Ressourcenadapter oder den IBM WebSphere MQ-Messaging-Provider von WebSphere Application Server 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 WebSphere 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 WebSphere MQ Classes for JMS

Eine WebSphere MQ Classes for JMS-Anwendung kann entweder im Client- oder im Bindungsmodus eine Verbindung zu einem Warteschlangenmanager herstellen. Im Clientmodus verbindet sich WebSphere MQ Classes for JMS über TCP/IP mit dem Warteschlangenmanager. Im Bindungsmodus stellt WebSphere MQ Classes for JMS über JNI (Java Native Interface) eine direkte Verbindung zum Warteschlangenmanager her.

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 von WebSphere 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 von WebSphere 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 WebSphere MQ Classes for JMS-Anwendung auf demselben System wie der Warteschlangenmanager ausgeführt werden, es ist aber auch die Ausführung auf einem anderen System möglich. In jedem Fall verbindet sich WebSphere MQ Classes for JMS über TCP/IP mit dem Warteschlangenmanager.

Bindungsmodus

Um sich im Bindungsmodus mit einem Warteschlangenmanager zu verbinden, muss eine WebSphere MQ Classes for JMS-Anwendung auf demselben System wie der Warteschlangenmanager ausgeführt werden.

WebSphere MQ Classes for JMS stellt über JNI (Java Native Interface) eine direkte Verbindung zum Warteschlangenmanager her. Damit der Bindungstransport verwendet werden kann, muss WebSphere MQ Classes for JMS in einer Umgebung ausgeführt werden, die auf die Bibliotheken von Java Native Interface zugreifen kann; weitere Informationen hierzu finden Sie im Abschnitt „JNI-Bibliotheken (Java Native Interface) konfigurieren” auf Seite 768.

WebSphere MQ Classes for JMS unterstützt 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 WebSphere 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 222.

Zur Verwendung des Bindungstransports muss die verwendete Java Runtime Environment die ID des codierten Zeichensatzes (CCSID) des Warteschlangenmanagers unterstützen, zu dem die WebSphere MQ -Klassen für JMS eine Verbindung herstellen.

Details dazu, wie Sie ermitteln können, welche CCSIDs von einer Java Runtime Environment unterstützt werden, finden Sie in WebSphere MQ FDC mit der Testmonitor-ID 21, die generiert wird, wenn die WebSphere MQ V7 -Klassen für Java oder WebSphere MQ V7 -Klassen für JMS verwendet werden.

Bindungen, dann Clientmodus

Dies ist die Standardeinstellung. Beim Verbinden mit einem Warteschlangenmanager in diesem Modus versucht eine WebSphere MQ Classes for JMS-Anwendung, die Verbindung im Bindungsmodus herzustellen. Dafür ist es erforderlich, dass sich der Warteschlangenmanager auf derselben Maschine wie die Anwendung befindet. Sollte die Verbindung nicht erfolgreich sein, versucht die Anwendung, die Verbindung im Clientmodus herzustellen. Dabei kann sich der Warteschlangenmanager auf derselben Maschine wie die Anwendung oder auf einer anderen Maschine befinden.

Warteschlangenmanager so konfigurieren, dass sich WebSphere MQ Classes for JMS-Anwendungen im Clientmodus verbinden können

Um Warteschlangenmanager so zu konfigurieren, dass WebSphere MQ Classes for JMS-Anwendungen Verbindungen im Clientmodus herstellen können, müssen Sie eine Serververbindungskanaldefinition erstellen und einen Listener starten.

Auf z/OS muss das Client Attachment Feature installiert sein.

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)
```

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.

Sie können die Definition eines Serververbindungskanals auch mit IBM WebSphere MQ Explorer erstellen, der unter Linux und Windowsausgeführt wird, oder mit den Operationen und Steuerkonsolen unter z/OS.

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.

Sie können auf allen Plattformen mit dem MQSC-Befehl START LISTENER einen Listener starten, bei z/OS müssen Sie jedoch zuerst mit dem MQSC-Befehl DEFINE LISTENER einen Listener erstellen. Sehen Sie sich das folgende Beispiel an:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

Auf UNIX-, Linux- und Windows-Systemen können Sie wie im folgenden Beispiel zum Starten eines Listeners auch den Steuerbefehl **runmqlsr** verwenden:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

In diesem Befehl steht *Warteschlangenmanagername* für den Namen Ihres Warteschlangenmanagers.

Zum Starten eines Listeners können Sie auch WebSphere MQ Explorer verwenden. Dieses Programm wird unter Linux und Windows ausgeführt. Sie haben aber auch die Möglichkeit, die Operationen und Bedienfelder unter z/OS zu nutzen.

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.

Der Punkt-zu-Punkt-Installationsprüftest für WebSphere MQ Classes for JMS

Mit WebSphere 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 von verwalteten Objekten finden Sie im Abschnitt „JMS-Objekttypen” auf Seite 997.

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. Das Script wird auf UNIX and Linux -Systemen als IVTRun und unter Windowsals IVTRun.bat bezeichnet und befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses von WebSphere 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, richten Sie zuerst den WS-Manager wie in „Beispielprogramme vorbereiten und ausführen” auf Seite 115beschrieben ein. Beachten Sie, dass der

zu verwendende Kanal standardmäßig **SYSTEM.DEF.SVRCONN** und die zu verwendende Warteschlange **SYSTEM.DEFAULT.LOCAL.QUEUE**ist. Geben Sie dann den folgenden Befehl ein:

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

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. Daher finden Sie weitere Informationen zu diesem Parameter und seinen gültigen Werten in der Beschreibung der Eigenschaft PROVIDERVERSION im Abschnitt Eigenschaften von IBM WebSphere 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 Traceverarbeitung wird eingeschaltet. Standardmäßig ist die Traceverarbeitung ausgeschaltet.

Ein erfolgreicher Test erstellt eine Ausgabe, die der folgenden Beispielausgabe ähnelt:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again
```

```
Got message
  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f6d627720202020202001edb14620005e03
  JMSTimestamp:     1187170264000
  JMSCorrelationID: null
  JMSDestination:   queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXUserID: mwhite
    JMS_IBM_Encoding: 273
    JMS_IBM_PutApplType: 28
    JMSXAppID: WebSphere 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 WebSphere 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 WebSphere MQ Classes for JMS” auf Seite 760. Informationen zur Konfiguration des WebSphere MQ-JMS-Verwaltungstools finden Sie im Abschnitt „JMS-Verwaltungstool konfigurieren” auf Seite 993.

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 hat das Script den Namen 'IVTSetup' und auf Windows-Systemen den Namen 'IVTSetup.bat'; es ist im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ-Klassen für JMS enthalten. Geben Sie folgenden Befehl ein, um das Script auszuführen:

```
IVTSetup
```

Das Script ruft das WebSphere 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 WebSphere MQ-JMS-Verwaltungstool oder mit WebSphere MQ Explorer ändern. Informationen zur Verwendung des WebSphere MQ JMS-Verwaltungstools erhalten Sie unter „Einsatz des WebSphere MQ JMS-Verwaltungstools” auf Seite 992. Informationen zur Verwendung von WebSphere MQ Explorer finden Sie in der bei WebSphere MQ Explorer bereitgestellten Hilfe.

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 Traceverarbeitung wird eingeschaltet. Standardmäßig ist die Traceverarbeitung ausgeschaltet.

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 hat das Script den Namen 'IVTTidy' und auf Windows-Systemen den Namen 'IVTTidy.bat'; es ist im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ-Klassen für JMS enthalten.

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, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden” auf Seite 766 beschrieben ordnungsgemäß festgelegt ist.
- Der Test kann mit der folgenden Nachricht fehlschlagen:

  ```
  Failed to connect to queue manager 'qmgr' with connection mode 'connMode'
  and host name 'hostname'
  ```

  Außerdem ist dieser Nachricht der Ursachencode 2059 zugeordnet. Die Variablen in der Nachricht haben folgende Bedeutungen:

  qmgr
  Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests versucht, eine Verbindung herzustellen. Diese Nachrichteneinfügung ist leer, wenn das Programm des Installationsprüftests versucht, sich im Bindungsmodus mit dem Standardwarteschlangenmanager zu verbinden.

connMode
Der Verbindungsmodus, entweder `Bindings` oder `Client`.

Hostname
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

Diese Nachricht bedeutet, dass der Warteschlangenmanager, zu dem das Programm des Installationsprüftests gerade eine Verbindung herstellen möchte, nicht verfügbar ist. Prüfen Sie, ob der Warteschlangenmanager aktiv ist, und falls das Programm des Installationsprüftests versucht, sich mit dem Standardwarteschlangenmanager zu verbinden, stellen Sie sicher, dass der Warteschlangenmanager als Standardwarteschlangenmanager für Ihr System definiert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

Diese Nachricht bedeutet, dass die Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE nicht in dem Warteschlangenmanager vorhanden ist, mit dem das Programm des Installationsprüftests verbunden ist. Falls die Warteschlange vorhanden ist, ist es auch möglich, dass das Programm des Installationsprüftests die Warteschlange nicht öffnen kann, weil sie nicht für das Einreihen und Abrufen von Nachrichten aktiviert wurde. Prüfen Sie, ob die Warteschlange vorhanden und für das Einreihen und Abrufen von Nachrichten aktiviert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Unable to bind to object
```

Diese Nachricht bedeutet, dass eine Verbindung mit dem LDAP-Server besteht, allerdings ist der LDAP-Server nicht ordnungsgemäß konfiguriert. Entweder ist der LDAP-Server nicht zum 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. Weitere Informationen finden Sie im Artikel „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

Der Publish/Subscribe-Installationsprüftest für WebSphere MQ Classes for JMS

Mit WebSphere 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 von verwalteten Objekten finden Sie im Abschnitt „JMS-Objekttypen” auf Seite 997.

Publish/Subscribe-Installationsprüftest ohne Verwendung von JNDI

Bei diesem Test erstellt und konfiguriert das Programm des Installationsprüftests alle von ihm benötigten Objekte dynamisch zur Laufzeit und verwendet kein JNDI.

Für die Ausführung des Programms des Installationsprüftests wird ein Script zur Verfügung gestellt. Das Script heißt PSIVTRun auf UNIX and Linux -Systemen und PSIVTRun.bat auf Windowsund befindet sich im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ -Klassen für 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 „Beispielprogramme vorbereiten und ausführen" auf Seite 115 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 nur dann relevant, wenn der Parameter -v eine Warteschlangenmanagerversion kleiner als 7 angibt und Sie WebSphere Event Broker oder WebSphere Message Broker als Publish/Subscribe-Broker verwenden.

-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 WebSphere 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 Traceverarbeitung wird eingeschaltet. Standardmäßig ist die Traceverarbeitung ausgeschaltet.

Ein erfolgreicher Test erstellt eine Ausgabe, die der folgenden Beispielausgabe ähnelt:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 7.0
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:414d5120514d5f6d62772020202020201edb14620006706
  JMSTimestamp:     1187182520203
  JMSCorrelationID: ID:414d5120514d5f6d62772020202020201edb14620006704
  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: 414D5143514D5F6D62772020202020201EDB14620006601
    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 WebSphere 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 WebSphere MQ Classes for JMS” auf Seite 760. Informationen zur Konfiguration des WebSphere MQ-JMS-Verwaltungstools finden Sie im Abschnitt „JMS-Verwaltungstool konfigurieren” auf Seite 993.

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 hat das Script den Namen 'IVTSetup' und auf Windows-Systemen den Namen 'IVTSetup.bat'; es ist im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ-Klassen für JMS enthalten. Geben Sie folgenden Befehl ein, um das Script auszuführen:

```
IVTSetup
```

Das Script ruft das WebSphere 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 WebSphere Event Broker oder WebSphere Message Broker anstelle der eingebetteten Publish/Subscribe-Funktion verwendet, müssen Sie die entsprechenden Eigenschaften des MQTopicConnectionFactory-Objekts mit dem WebSphere MQ-JMS-Verwaltungstool oder mit WebSphere MQ Explorer ändern. Informationen zur Verwendung des WebSphere MQ JMS-Verwaltungstools erhalten Sie unter „Einsatz des WebSphere MQ JMS-Verwaltungstools” auf Seite 992. Informationen zur Verwendung von WebSphere MQ Explorer finden Sie in der bei WebSphere MQ Explorer bereitgestellten Hilfe.

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 Traceverarbeitung wird eingeschaltet. Standardmäßig ist die Traceverarbeitung ausgeschaltet.

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 hat das Script den Namen 'IVTTidy' und auf Windows-Systemen den Namen 'IVTTidy.bat'; es ist im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ-Klassen für JMS enthalten.

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, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden” auf Seite 766 beschrieben ordnungsgemäß festgelegt ist.
- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Failed to connect to queue manager 'qmgr' with
connection mode 'connMode' and host name 'hostname'
```

Außerdem ist dieser Nachricht der Ursachencode 2059 zugeordnet. Die Variablen in der Nachricht haben folgende Bedeutungen:

qmgr
Der Name des Warteschlangenmanagers, zu dem das Programm des Installationsprüftests versucht, eine Verbindung herzustellen. Diese Nachrichteneinfügung ist leer, wenn das Programm des Installationsprüftests versucht, sich im Bindungsmodus mit dem Standardwarteschlangenmanager zu verbinden.

connMode
Der Verbindungsmodus, entweder `Bindings` oder `Client`.

Hostname
Der Hostname oder die IP-Adresse des Systems, auf dem der Warteschlangenmanager ausgeführt wird.

Diese Nachricht bedeutet, dass der Warteschlangenmanager, zu dem das Programm des Installationsprüftests gerade eine Verbindung herstellen möchte, nicht verfügbar ist. Prüfen Sie, ob der Warteschlangenmanager aktiv ist, und falls das Programm des Installationsprüftests versucht, sich mit dem Standardwarteschlangenmanager zu verbinden, stellen Sie sicher, dass der Warteschlangenmanager als Standardwarteschlangenmanager für Ihr System definiert ist.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
Unable to bind to object
```

Diese Nachricht bedeutet, dass eine Verbindung mit dem LDAP-Server besteht, allerdings ist der LDAP-Server nicht ordnungsgemäß konfiguriert. Entweder ist der LDAP-Server nicht zum Speichern von Java-Objekten konfiguriert oder die Berechtigungen für die Objekte oder das Suffix sind nicht korrekt. Falls Sie in dieser Situation weitere Hilfe benötigen, lesen Sie die Dokumentation Ihres LDAP-Servers.

- Der Test kann mit der folgenden Nachricht fehlschlagen:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

Diese Nachricht bedeutet, dass der Warteschlangenmanager nicht ordnungsgemäß eingerichtet ist, um eine Clientverbindung von Ihrem System zu akzeptieren. Weitere Informationen finden Sie unter „Beispielprogramme vorbereiten und ausführen” auf Seite 115.

Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter

Das Programm des Installationsprüftests (Installation Verification Test, IVT) wird als EAR-Datei bereitgestellt. Um das Programm verwenden zu können, müssen Sie es implementieren und einige Objekte als JCA-Ressourcen definieren.

Das Programm des Installationsprüftests (IVT) wird als Unternehmensarchivdatei (EAR-Datei) mit dem Namen 'wmq.jmsra.ivt.ear' bereitgestellt. Diese Datei wird mit WebSphere MQ Classes for JMS in demselben Verzeichnis installiert wie die RAR-Datei 'wmq.jmsra.rar' des WebSphere MQ-Ressourcenadapters. Informationen zur Installationsposition dieser Dateien finden Sie im Abschnitt „Installation des WebSphere MQ-Ressourcenadapters” auf Seite 778.

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 WebSphere MQ-Warteschlange gesendet bzw. aus dieser empfangen werden kann. Optional können Sie das IVT-Programm verwenden, um zu prüfen, ob der WebSphere MQ-Ressourcenadapter ordnungsgemäß für die Unterstützung der dezentralen Transaktionsverarbeitung konfiguriert wurde.

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. Sie können die Eigen-

schaften der Objekte frei wählen. Bei der folgenden Gruppe von Eigenschaften handelt es sich um ein einfaches Beispiel:

ConnectionFactory, Objekt

```
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1414
queueManager:         ExampleQM
transportType:        CLIENT
```

Queue-Objekt

```
baseQueueManagerName:  ExampleQM
baseQueueName:         TEST.QUEUE
```

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 starten, indem Sie eine URL im folgenden Format in Ihren Web-Browser eingeben:

```
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 122 auf Seite 833 ist die erste Seite des IVT-Programms dargestellt.

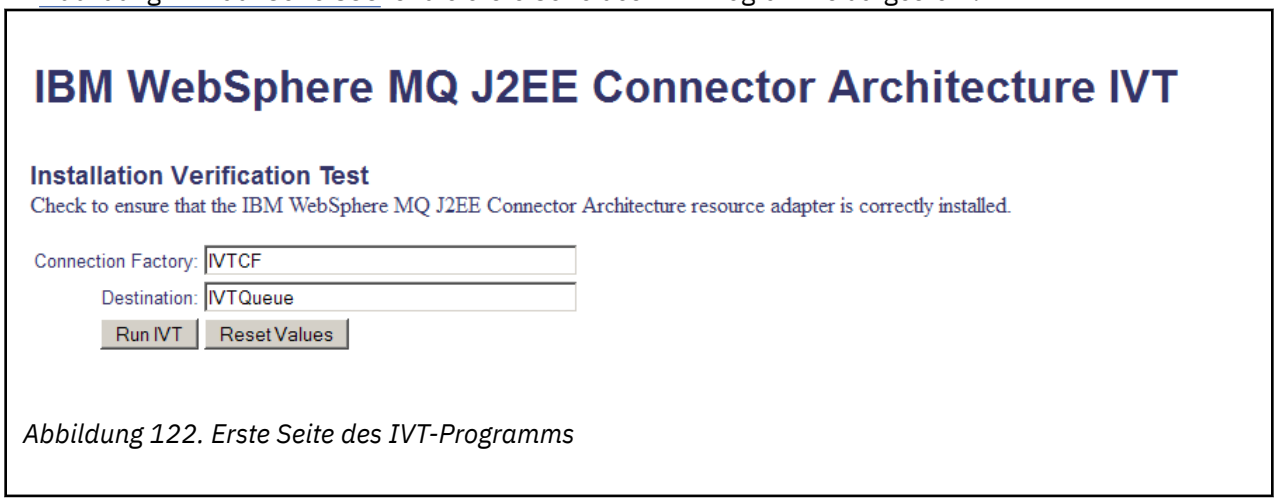


Abbildung 122. Erste Seite des IVT-Programms

Wenn Sie den Test ausführen möchten, klicken Sie auf **Run IVT** (IVT ausführen).Abbildung 123 auf Seite 834 zeigt die Seite, die angezeigt wird, wenn der IVT erfolgreich ist.

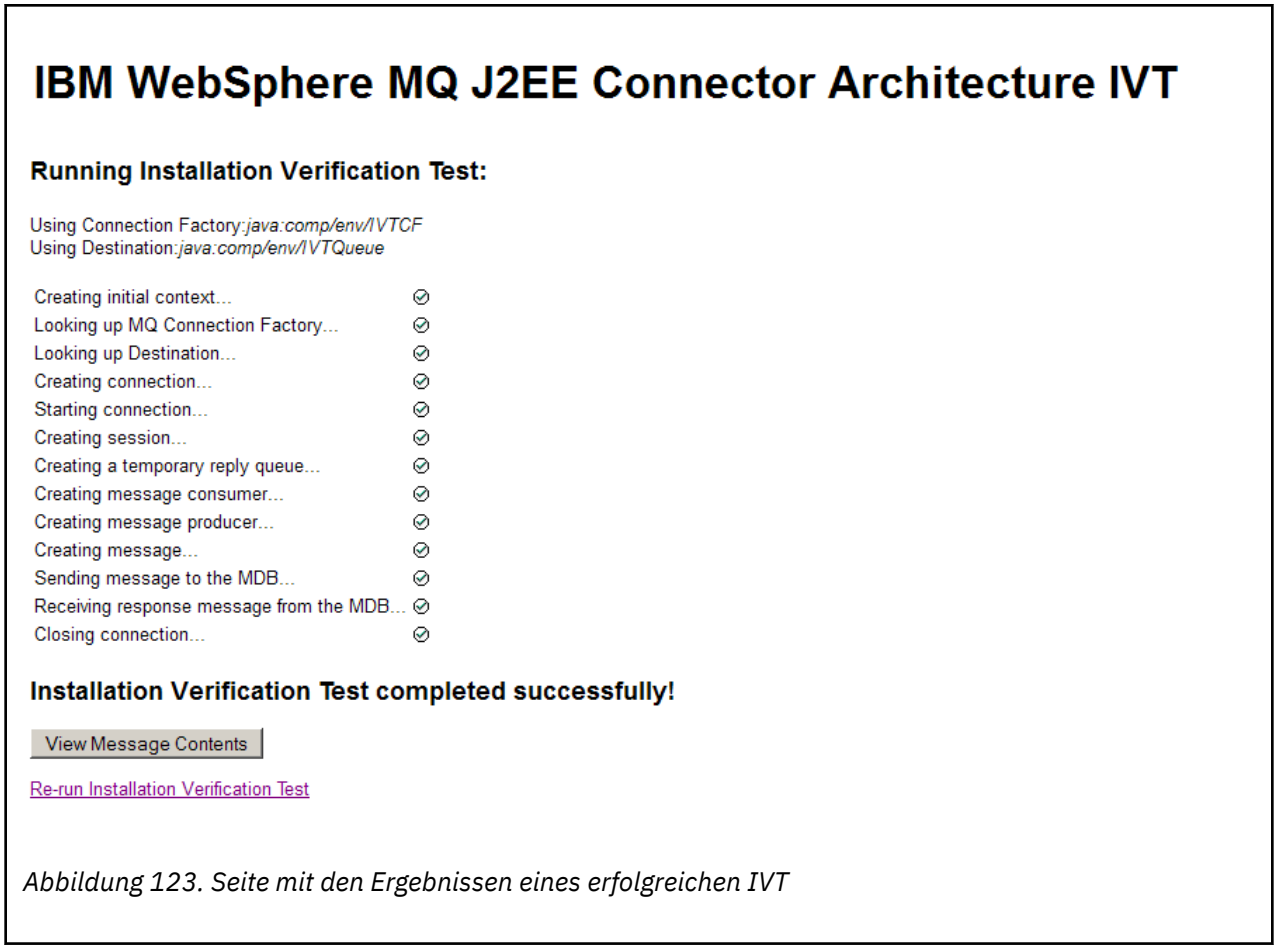


Abbildung 123. Seite mit den Ergebnissen eines erfolgreichen IVT

Wenn der IVT fehlschlägt, wird eine Seite angezeigt, die derjenigen in Abbildung 124 auf Seite 835 ähnelt. Wenn Sie weitere Informationen zur Fehlerursache erhalten möchten, klicken Sie auf **View Stack Trace** (Stack-Trace anzeigen).

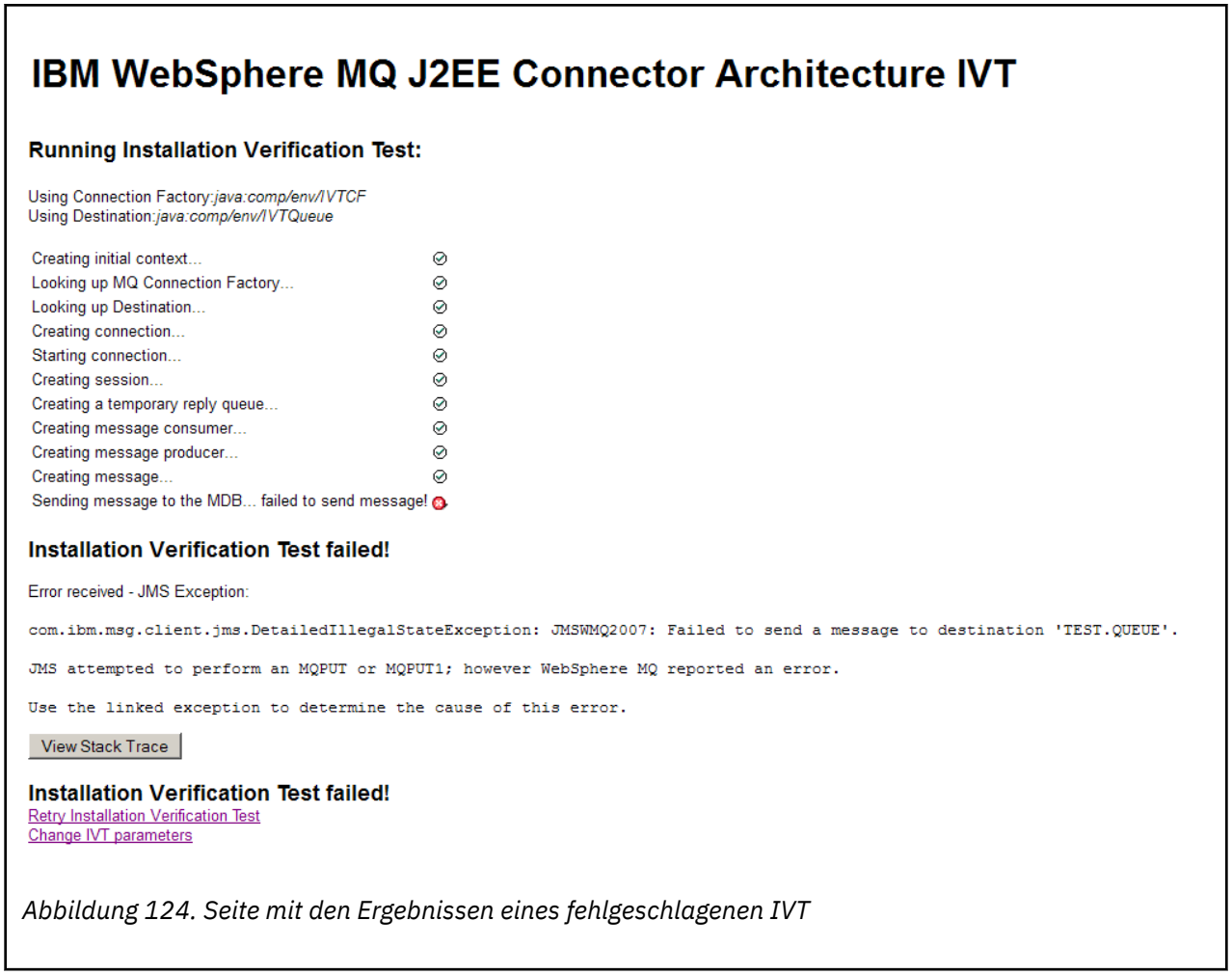


Abbildung 124. Seite mit den Ergebnissen eines fehlgeschlagenen IVT

Ausführliche Anweisungen und Informationen zu Dienstprogrammscripts, die zum Implementieren der IVT-Anwendung auf JBoss -und WAS CE-Anwendungsservern bereitgestellt werden, finden Sie unter:

Zugehörige Tasks

„MQ-Ressourcenadapter unter WAS CE installieren und testen” auf Seite 835
Installation des IBM WebSphere MQ-Ressourcenadapters und Ausführung der IVT-Anwendung (IVT = Installation Verification Test, Installationsprüftest) in WebSphere Application Server CE.

„Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen” auf Seite 839
Nachdem Sie den IBM WebSphere MQ -Ressourcenadapter unter JBoss AS 5.1 oder 6 installiert haben, können Sie die Installation des Ressourcenadapters testen, indem Sie die IVT-Anwendung (Installation Verification Test, Installationsprüftest) installieren und ausführen.

MQ-Ressourcenadapter unter WAS CE installieren und testen

Installation des IBM WebSphere MQ-Ressourcenadapters und Ausführung der IVT-Anwendung (IVT = Installation Verification Test, Installationsprüftest) in WebSphere Application Server CE.

Vorbereitende Schritte

Diese Task setzt voraus, dass Sie über einen aktiven WebSphere Application Server CE-Server verfügen und mit den Standardverwaltungstasks für den Server vertraut sind. Darüber hinaus wird vorausgesetzt, dass Sie über eine IBM WebSphere MQ-Installation auf Ihrem lokalen System verfügen und mit den diesbezüglichen Standardverwaltungsaufgaben vertraut sind.

Wenn Sie den Ressourcenadapter für die Verbindung zu einem IBM WebSphere MQ -Client verwenden und verteilte XA-Transaktionen ausführen müssen, müssen Sie die zusätzlichen Schritte ausführen, die mit **Nur Client XA**gekennzeichnet sind.

1. Erstellen Sie einen Warteschlangenmanager mit dem Namen `ExampleQM` und richten Sie ihn gemäß der Beschreibung im Abschnitt „Beispielprogramme vorbereiten und ausführen" auf Seite 115 ein. Beachten Sie dabei, dass der Listener an Port 1414 gestartet werden muss, der Kanal `SYSTEM.DEF.SVRCONN` verwendet werden muss und durch die IVT-Anwendung die Warteschlange `TEST.QUEUE` verwendet werden muss. Außerdem müssen der Modellwarteschlange `SYSTEM.DEFAULT.MODEL.QUEUE` die Berechtigungen DSP und PUT erteilt werden, damit diese Anwendung eine temporäre Antwortwarteschlange erstellen kann. Wenn Sie einen anderen Warteschlangenmanager, andere Verbindungsdetails oder eine andere Warteschlange verwenden möchten, finden Sie die entsprechenden Informationen im Abschnitt „IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren" auf Seite 837.
2. Rufen Sie die Ressourcenadapterdatei (`wmq.jmsra.rar`), die IVT-Anwendung (`wmq.jmsra.ivt.ear`) sowie `WAS_CE_jmsra_deployment_plan.xml` und `WAS_CE_jmsra_ivt_deployment_plan.xml deployment plan filesab`. Details zur Position dieser Dateien finden Sie in „Installation des WebSphere MQ-Ressourcenadapters" auf Seite 778.

Eine Beschreibung der Verbindungen im Bindungs- und im Clientmodus finden Sie im Abschnitt „Verbindungsmodi für WebSphere MQ Classes for JMS" auf Seite 822.

Wenn Sie eine andere Warteschlange, einen anderen Warteschlangenmanager, Port, Host oder Kanal oder den Bindungsmodus statt des Clientmodus verwenden möchten, finden Sie die entsprechenden Informationen im Abschnitt „IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren" auf Seite 837.

Vorgehensweise

1. **Nur Client XA**: Bearbeiten Sie Ihre Kopie der Datei `WAS_CE_jmsra_deployment_plan.xml`.
   a) Suchen Sie die Verbindungsdefinition jms/ivt/IVTCF und modifizieren Sie sie so, dass die Verbindungsfactory XA-Transaktionen unterstützt.
      i) Kommentieren Sie den NonXA-Abschnitt aus:

```
<conn:xa-transaction>
```

      ii) Entfernen Sie das Kommentarzeichen beim XA-Konfigurationsabschnitt:

```
<conn:xa-transaction>
    <conn:transaction-caching/>
</conn:xa-transaction>
```

   b) Speichern Sie Ihre Änderungen.
2. Optional: **Nur Client XA**: Ändern Sie den Assemblierungsdeskriptor der MDB so, dass Transaktionen erforderlich sind. Dies zwingt die MDB im IVT, an einer XA-Transaktion teilzunehmen, obwohl die IVT-Anwendung noch ohne diese Änderung funktioniert.
   a) Öffnen Sie die Datei `wmq.jmsra.ivt.ear`.
   b) Öffnen Sie die darin enthaltene Datei `WMQ_IVT_MDB.jar`.
   c) Bearbeiten Sie die Datei `META-INF/ejb-jar.xml`.
      i) Kommentieren Sie die Zeile innerhalb des Assemblierungsdeskriptors aus oder löschen Sie sie:

```
<trans-attribute>NotSupported</trans-attribute>
```

      ii) Entfernen Sie das Kommentarzeichen bei der Zeile innerhalb des Assemblierungsdeskriptors:

```
<trans-attribute>Required</trans-attribute>
```

iii) Speichern Sie die Änderungen und aktualisieren Sie die Datei in der JAR-Datei `WMQ_IVT_MDB.jar`.

iv) Aktualisieren Sie die Datei 'wmq.jmsra.ivt.ear' mit der geänderten JAR-Datei `WMQ_IVT_MDB.jar`.

3. Implementieren Sie mit der modifizierten Implementierungsplandatei den Ressourcenadapter auf Ihrem Server.
   a) Um dies auf der Befehlszeile auszuführen, geben Sie den folgenden WAS CE-Befehl ein:

```
deploy -u system -p manager deploy wmq.jmsra.rar WAS_CE_jmsra_deployment_plan.xml
```

   b) Rufen Sie in der Webverwaltungsschnittstelle **Anwendungen > Implementierer** auf.

      i) Legen Sie als Archiv die Datei wmq.jmsra.rar fest.

      ii) Legen Sie als Plan die Datei WAS_CE_jmsra_deployment_plan.xml fest.

      iii) Stellen Sie sicher, dass 'Anwendung nach Installation starten' ausgewählt ist.

      iv) Klicken Sie auf **Install** (Installieren).

4. Implementieren Sie mit dem bereitgestellten Implementierungsplan die IVT-Anwendung auf Ihrem Server.
   a) Um dies auf der Befehlszeile auszuführen, geben Sie den folgenden WAS CE-Befehl ein:

```
deploy -u system -p manager deploy wmq.jmsra.ivt.ear WAS_CE_jmsra_ivt_deployment_plan.xml
```

   b) Rufen Sie in der Webverwaltungsschnittstelle **Anwendungen > Implementierer** auf.

      i) Setzen Sie als **Archiv** die Datei `wmq.jmsra.ivt.ear`.

      ii) Setzen Sie als **Plan** die Datei `WAS_CE_jmsra_ivt_deployment_plan.xml`.

      iii) Stellen Sie sicher, dass die Option **Start application after install** (Anwendung nach Installation starten) ausgewählt ist.

      iv) Klicken Sie auf **Install** (Installieren).

5. Führen Sie die IVT-Anwendung aus. Weitere Informationen finden Sie unter „Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter” auf Seite 832. Die Standard-URL für WAS CE ist `http://localhost:8080/WMQ_IVT/`.

IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren

Wenn Sie eine andere Warteschlange, einen anderen Warteschlangenmanager, einen anderen Port, einen anderen Host, einen anderen Kanal oder den Bindungsmodus anstelle des Clientmodus verwenden möchten, müssen Sie die IVT-Anwendung und die zugehörigen Scripts in WebSphere Application Server CE ändern, bevor Sie den Ressourcenadapter oder die IVT-Anwendung implementieren.

Informationen zu diesem Vorgang

Soll eine andere als die unter „MQ-Ressourcenadapter unter WAS CE installieren und testen” auf Seite 835 dargestellte Konfiguration implementiert werden, d.h., soll eine andere Warteschlange oder ein anderer Warteschlangenmanager, Port, Host oder Kanal, oder der Clientmodus anstelle des Bindungsmodus verwendet werden, müssen Sie vor einer Implementierung des Ressourcenadapters oder der IVT-Anwendung folgende Schritte ausführen:

Vorgehensweise

1. Wenn Sie für die IVT-Anwendung einen anderen Warteschlangenmanager und eine andere Warteschlange angeben möchten, dann legen Sie die Werte für den Warteschlangenmanager und die Warteschlange in `WAS_CE_jmsra_deployment_plan.xml` fest. Details finden Sie im Abschnitt „Werte für Warteschlangenmanager und Warteschlange festlegen” auf Seite 838.

2. Soll in der Konfiguration der MDB (Message-driven Bean) ein anderer Warteschlangenmanager und eine andere Warteschlange angegeben werden, müssen Sie für den Warteschlangenmanager und die Warteschlange, die Sie verwenden, in `WAS_CE_jmsra_ivt_deployment_plan.xml` entsprechende Werte setzen; weitere Informationen finden Sie unter „Werte für die MDB-Konfiguration festlegen" auf Seite 838.
3. Wenn Sie den Ressourcenadapter so konfigurieren, dass er eine Verbindung zu IBM WebSphere MQ im Bindungsmodus herstellt, müssen die JNI-Bibliotheken sich im Systempfad oder im Pfad für WAS CE befinden. Weitere Informationen finden Sie im Abschnitt „MQ-Ressourcenadapter unter WAS CE installieren und testen" auf Seite 835.
4. Wurde der Ressourcenadapter bereits implementiert, können Sie ihn mit dem folgenden Befehl mit dem geänderten Implementierungsplan erneut implementieren, um die Einstellungen zu ändern:

   ```
   deploy --user system --password manager redeploy wmq.jmsra.rar
   WAS_CE_jmsra_deployment_plan.xml
   ```

Nächste Schritte

Setzen Sie die Implementierung des Ressourcenadapters wie unter „MQ-Ressourcenadapter unter WAS CE installieren und testen" auf Seite 835 beschrieben fort.

Werte für Warteschlangenmanager und Warteschlange festlegen

Erläuterung der Vorgehensweise beim Festlegen der Werte für den Warteschlangenmanager und die Warteschlange, die Sie verwenden, in `WAS_CE_jmsra_deployment_plan.xml`.

Vorgehensweise

Legen Sie in `WAS_CE_jmsra_deployment_plan.xml` die Werte für den Warteschlangenmanager und die Warteschlange fest, die Sie für die IVT-Anwendung verwenden.

Für die Verbindungsdefinition jms/ivt/IVTCF:

1. Legen Sie als Wert des queueManager-Elements den Namen Ihres Warteschlangenmanagers fest.
2. Wenn Sie eine Clientverbindung verwenden, dann legen Sie den Wert der verschiedenen Clientverbindungselemente so fest, dass sie einer Verbindung zu Ihrem Warteschlangenmanager entsprechen.
3. Wenn Sie eine Bindungsverbindung verwenden:
   a. Legen Sie den Wert des transportType-Elements auf BINDINGS fest.
   b. Kommentieren Sie die verschiedenen Clientverbindungselemente aus oder löschen Sie sie.
4. Legen Sie für das Nachrichtenziel jms/ivt/IVTQueue als Wert des baseQueueName-Elements den Namen der Warteschlange fest, die Sie für die IVT-Anwendung erstellt haben.
5. Speichern Sie Ihre Änderungen.

Werte für die MDB-Konfiguration festlegen

Erläutert, wie Werte für die MDB-Konfiguration in `WAS_CE_jmsra_deployment_plan.xml`festgelegt werden.

Vorgehensweise

Legen Sie in `WAS_CE_jmsra_ivt_deployment_plan.xml` die Werte für den Warteschlangenmanager und die Warteschlange fest, die Sie in der Konfiguration für die MDB verwenden.

Für die Message-driven Bean WMQ_IVT_MDB:

1. Legen Sie als Wert des queueManager-Elements den Namen Ihres Warteschlangenmanagers fest.
2. Wenn Sie eine Clientverbindung verwenden, dann legen Sie den Wert der verschiedenen Clientverbindungselemente so fest, dass sie einer Verbindung zu Ihrem Warteschlangenmanager entsprechen.
3. Wenn Sie eine Bindungsverbindung verwenden:
   a. Legen Sie den Wert des transportType-Elements auf BINDINGS fest.

b. Kommentieren Sie die verschiedenen Clientverbindungselemente aus oder löschen Sie sie.

4. Speichern Sie Ihre Änderungen.

Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen

Nachdem Sie den IBM WebSphere MQ -Ressourcenadapter unter JBoss AS 5.1 oder 6 installiert haben, können Sie die Installation des Ressourcenadapters testen, indem Sie die IVT-Anwendung (Installation Verification Test, Installationsprüftest) installieren und ausführen.

Vorbereitende Schritte

Wichtig: Diese Anweisungen gelten für JBoss AS 5.1 und 6 und sind für JBoss AS 7 nicht gültig.

Informationen zur Installation des Ressourcenadapters in JBoss EAP 6.3finden Sie unter „Ressourcenadapter in JBoss EAP 6.3 installieren und testen” auf Seite 841.

Diese Task setzt voraus, dass Sie über einen aktiven JBoss -Server verfügen und mit den Standardverwaltungstasks für ihn vertraut sind. Darüber hinaus wird vorausgesetzt, dass Sie über eine IBM WebSphere MQ-Installation auf Ihrem lokalen System verfügen und mit den diesbezüglichen Standardverwaltungsaufgaben vertraut sind.

Wenn Sie den Ressourcenadapter für die Verbindung zu einem IBM WebSphere MQ -Client verwenden und verteilte XA-Transaktionen ausführen müssen, müssen Sie die zusätzlichen Schritte ausführen, die mit **Nur Client XA**gekennzeichnet sind. Eine Beschreibung der Verbindungen im Bindungs- und im Clientmodus finden Sie im Abschnitt „Verbindungsmodi für WebSphere MQ Classes for JMS” auf Seite 822.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager des Namens 'ExampleQM' und konfigurieren Sie ihn wie unter „Beispielprogramme vorbereiten und ausführen” auf Seite 115 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.

   Wenn Sie einen anderen Warteschlangenmanager, andere Verbindungsdetails oder eine andere Warteschlange verwenden möchten, finden Sie die entsprechenden Informationen im Abschnitt „IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren” auf Seite 837.

2. Besorgen Sie sich die Ressourcenadapterdatei (`wmq.jmsra.rar`), die IVT-Anwendung (`wmq.jmsra.ivt.ear`) und die Datei `jboss-jmsra-ds.xml`.

   Angaben zu dem Pfad, in dem diese Dateien enthalten sind, finden Sie unter „Installation des WebSphere MQ-Ressourcenadapters” auf Seite 778.

3. **Nur Client XA**: Bearbeiten Sie die Datei `jboss-jmsra-ds.xml` , um XA-Transaktionen in der Verbindungsfactory zu aktivieren.

   a) Setzen Sie die Zeile in der Verbindungsfactory-Definition `<local-transaction/>`auf Kommentar oder löschen Sie sie.

   b) Entfernen Sie das Kommentarzeichen aus der Zeile in der Verbindungsfactory-Definition `<xa-transaction/>`.

   c) Speichern Sie Ihre Änderungen.

4. **Nur Client XA**: (optional) Modifizieren Sie den Assemblierungsdeskriptor der MDB so, dass Transaktionen gefordert werden. Dies zwingt die MDB im IVT, an einer XA-Transaktion teilzunehmen, obwohl die IVT-Anwendung noch ohne diese Änderung funktioniert.

   a) Öffnen Sie die Datei `wmq.jmsra.ivt.ear`.

b) Öffnen Sie die darin enthaltene Datei `WMQ_IVT_MDB.jar`.

c) Bearbeiten Sie `META-INF/ejb-jar.xml`:

i) Kommentieren Sie die Zeile innerhalb des Assemblierungsdeskriptors aus oder löschen Sie sie:

```
<trans-attribute>NotSupported</trans-attribute>
```

ii) Entfernen Sie das Kommentarzeichen bei der Zeile innerhalb des Assemblierungsdeskriptors:

```
<trans-attribute>Required</trans-attribute>
```

iii) Speichern Sie die Änderungen und aktualisieren Sie die Datei in der JAR-Datei `WMQ_IVT_MDB.jar`.

iv) Aktualisieren Sie die Datei `wmq.jmsra.ivt.ear` mit der geänderten JAR-Datei `WMQ_IVT_MDB.jar`.

5. Implementieren Sie den Ressourcenadapter auf Ihrem Server, indem Sie die Datei `wmq.jmsra.rar` in das Verzeichnis `jboss/server/default/deploy` kopieren.
6. Erstellen Sie die für die IVT-Anwendung erforderlichen JMS-Ressourcen, indem Sie die Datei `jboss-jmsra-ds.xml` in das Verzeichnis `jboss/server/default/deploy` kopieren.
7. Implementieren Sie die IVT-Anwendung, indem Sie die Datei `wmq.jmsra.ivt.ear` in das Verzeichnis `jboss/server/default/deploy` kopieren.
8. Führen Sie die IVT-Anwendung aus. Weitere Informationen finden Sie in „Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter" auf Seite 832. Für JBosslautet die Standard-URL `http://localhost:8080/WMQ_IVT/`.

IVT-Anwendung in JBoss mit einer angepassten IBM WebSphere MQ -Umgebung implementieren

Wenn Sie bei der Installation des IBM WebSphere MQ -Ressourcenadapters in JBosseine andere Warteschlange, einen anderen Warteschlangenmanager, einen anderen Port, einen anderen Host, einen anderen Kanal oder den Bindungsmodus anstelle des Clientmodus verwenden wollen, müssen Sie zuerst die IVT-Anwendung und die zugehörigen Scripts in JBoss ändern, bevor Sie den Ressourcenadapter oder die IVT-Anwendung implementieren.

Informationen zu diesem Vorgang

Wichtig: Diese Anweisungen gelten nur für Java EE Version 6 und 5, nicht für Java EE Version 7. Die Verwendung dieser Anweisungen für JBoss Version 8 (WildFly) wird daher nicht unterstützt.

Soll eine andere als die unter „Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen" auf Seite 839 dargestellte Konfiguration implementiert werden, d.h., soll eine andere Warteschlange oder ein anderer Warteschlangenmanager, Port, Host oder Kanal, oder der Clientmodus anstelle des Bindungsmodus verwendet werden, müssen Sie vor einer Implementierung des Ressourcenadapters oder der IVT-Anwendung folgende Schritte ausführen:

Vorgehensweise

1. Wenn Sie für die IVT-Anwendung einen anderen Warteschlangenmanager und eine andere Warteschlange angeben möchten, dann legen Sie die Werte für den Warteschlangenmanager und die Warteschlange fest.

a) Für die Verbindungsdefinition jms/ivt/IVTCF:

i) Legen Sie als Wert der queueManager-Konfigurationseigenschaft den Namen Ihres Warteschlangenmanagers fest.

ii) Wenn Sie eine Clientverbindung verwenden, dann legen Sie den Wert der verschiedenen Clientverbindungselemente so fest, dass sie einer Verbindung zu Ihrem Warteschlangenmanager entsprechen.

iii) Wenn Sie eine Bindungsverbindung verwenden, dann legen Sie den Wert des transportType-Elements auf BINDINGS fest und kommentieren Sie dann die verschiedenen Clientverbindungselemente aus oder löschen Sie sie.

b) Legen Sie für die MBean jms/ivt/IVTQueue als Wert des baseQueueName-Elements den Namen der Warteschlange fest, die Sie für die IVT-Anwendung erstellt haben.

c) Speichern Sie Ihre Änderungen.

2. Soll in der Konfiguration für die MDB (Message-driven Bean) ein anderer Warteschlangenmanager und eine andere Warteschlange angegeben werden, müssen Sie die Konfiguration der MDB so ändern, dass eine Verbindung zu diesem Warteschlangenmanager und zu dieser Warteschlange hergestellt wird.

a) Öffnen Sie die Datei `wmq.jmsra.ivt.ear`.

b) Öffnen Sie die darin enthaltene Datei `WMQ_IVT_MDB.jar`.

c) Bearbeiten Sie `META-INF/ejb-jar.xml`:

i) Legen Sie als Wert der queueManager-Aktivierungskonfigurationseigenschaft den Namen Ihres Warteschlangenmanagers fest.

ii) Wenn Sie eine Clientverbindung verwenden, dann legen Sie den Wert der verschiedenen Aktivierungskonfigurationseigenschaften so fest, dass sie einer Verbindung zu Ihrem Warteschlangenmanager entsprechen.

iii) Wenn Sie eine Bindungsverbindung verwenden, dann legen Sie den Wert der transportType-Aktivierungskonfigurationseigenschaft auf BINDINGS fest und kommentieren Sie dann die verschiedenen Clientverbindungselemente aus oder löschen Sie sie.

d) Speichern Sie die Änderungen und aktualisieren Sie die Datei in der JAR-Datei `WMQ_IVT_MDB.jar`.

e) Aktualisieren Sie die Datei `wmq.jmsra.ivt.ear` mit der geänderten JAR-Datei `WMQ_IVT_MDB.jar`.

3. Wenn Sie den Ressourcenadapter so konfigurieren, dass er eine Verbindung zu IBM WebSphere MQ im Bindungsmodus herstellt, stellen Sie sicher, dass sich die JNI-Bibliotheken im Systempfad oder im Pfad für JBossbefinden. Details hierzu finden Sie unter „JNI-Bibliotheken (Java Native Interface) konfigurieren” auf Seite 768.

Nächste Schritte

Setzen Sie die Implementierung des Ressourcenadapters wie unter „Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen” auf Seite 839 beschrieben fort.

.

Ressourcenadapter in JBoss EAP 6.3 installieren und testen

Nach der Installation des IBM WebSphere MQ -Ressourcenadapters in JBoss Enterprise Application Platform (EAP) 6.3(entweder auf einem eigenständigen Server oder auf einem Server, der in einer verwalteten Domäne ausgeführt wird) können Sie die Installation des Ressourcenadapters testen, indem Sie die IVT-Anwendung (Installation Verification Test, Installationsprüftest) installieren und ausführen.

Informationen zu diesem Vorgang

Wichtig: Diese Anweisungen gelten nur für JBoss EAP 6.3 . Informationen zur Installation des Ressourcenadapters in JBoss AS 5.1 und 6 finden Sie in „Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen” auf Seite 839.

Diese Task setzt voraus, dass Sie über einen aktiven JBoss -Server verfügen und mit den Standardverwaltungstasks für ihn vertraut sind. Außerdem wird vorausgesetzt, dass Sie über eine IBM WebSphere MQ -Installation auf Ihrem lokalen System verfügen und mit der Standardverwaltung vertraut sind.

Vorgehensweise

1. Erstellen Sie einen Warteschlangenmanager des Namens 'ExampleQM' und konfigurieren Sie ihn wie unter „Beispielprogramme vorbereiten und ausführen” auf Seite 115 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.

   Wenn Sie einen anderen Warteschlangenmanager, andere Verbindungsdetails oder eine andere Warteschlange verwenden möchten, finden Sie die entsprechenden Informationen im Abschnitt „IVT-Anwendung in WAS CE mit einer angepassten MQ-Umgebung implementieren” auf Seite 837.
2. Besorgen Sie sich die Ressourcenadapterdatei (`wmq.jmsra.rar`) und die IVT-Anwendung (`wmq.jmsra.ivt.ear`).

   Angaben zu dem Pfad, in dem diese Dateien enthalten sind, finden Sie unter „Installation des WebSphere MQ-Ressourcenadapters” auf Seite 778.
3. Installieren Sie den Ressourcenadapter und testen Sie die Installation anschließend, indem Sie die IVT-Anwendung ausführen:

   - Wird der Ressourcenadapter auf einem Standalone-Server installiert, finden Sie entsprechende Informationen unter „Installation und Test auf einem Standalone-Server” auf Seite 842.
   - Wird der Ressourcenadapter auf einem in einer verwalteten Domäne eingesetzten Server installiert, finden Sie entsprechende Informationen unter „Installation und Test auf einem in einer verwalteten Domäne eingesetzten Server” auf Seite 843.

Installation und Test auf einem Standalone-Server

Nach der Installation des IBM WebSphere MQ -Ressourcenadapters unter JBoss EAP 6.3 auf einem eigenständigen Server können Sie die Installation des Ressourcenadapters testen, indem Sie die IVT-Anwendung (Installation Verification Test, Installationsprüftest) installieren und ausführen.

Informationen zu diesem Vorgang

Die Informationen in diesem Abschnitt gelten für die Installation und den Test des Ressourcenadapters auf einem Standalone-Server. Wird der Ressourcenadapter auf einem in einer verwalteten Domäne eingesetzten Server installiert, finden Sie entsprechende Informationen unter „Installation und Test auf einem in einer verwalteten Domäne eingesetzten Server” auf Seite 843.

Wichtig: Diese Anweisungen gelten nur für JBoss EAP 6.3 . Informationen zur Installation des Ressourcenadapters in JBoss AS 5.1 und 6 finden Sie in „Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen” auf Seite 839.

Vorgehensweise

1. Implementieren Sie den Ressourcenadapter auf Ihrem Server, indem Sie die Datei `wmq.jmsra.rar` in das Verzeichnis <*EAP_HOME*>`/standalone/deployments` kopieren.
2. Erstellen Sie die für die IVT-Anwendung erforderlichen JMS-Ressourcen, indem Sie die folgenden Einträge zum Abschnitt `<resource-adapters>` der Datei <*EAP_HOME*>`/standalone/configuration/standalone-full.xml` hinzufügen:

```
<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.ManagedConnectionFactoryImpl"
```

```
          jndi-name="java:jboss/jms/ivt/IVTCF"
          enabled="true"
          use-java-context="true"
          pool-name="IVTCF">
        <config-property name="port">
            1414
        </config-property>
        <config-property name="hostName">
            localhost
        </config-property>
        <config-property name="channel">
            SYSTEM.DEF.SVRCONN
        </config-property>
        <config-property name="transportType">
            CLIENT
        </config-property>
        <config-property name="queueManager">
            ExampleQM
        </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>
```

3. Fügen Sie den Startparametern des Anwendungsservers die folgenden Informationen hinzu:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Implementieren Sie die IVT-Anwendung, indem Sie die Datei `wmq.jmsra.ivt.ear` in das Verzeichnis <*EAP_HOME*>`/standalone/deployments`kopieren.
5. Starten Sie den Anwendungsserver.
6. Führen Sie die IVT-Anwendung aus.

   Weitere Informationen finden Sie im Abschnitt „Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter” auf Seite 832. Für JBosslautet die Standard-URL `http://localhost:8080/WMQ_IVT/`.

   Anmerkung: Die JNDI-Namen der für die Ausführung der IVT-Anwendung erforderlichen JMS-Ressourcen lauten wie folgt:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   Wenn Sie die IVT-Anwendung über die oben angegebene URL starten, geben Sie die JNDI-Namen dieser Ressourcen in die entsprechenden Felder ein und führen Sie dann die Anwendung aus, indem Sie auf **IVT ausführen**klicken.

Installation und Test auf einem in einer verwalteten Domäne eingesetzten Server

Nach der Installation des IBM WebSphere MQ -Ressourcenadapters in JBoss EAP 6.3 auf einem Server in einer verwalteten Domäne können Sie die Installation des Ressourcenadapters testen, indem Sie die IVT-Anwendung (IVT = Installation Verification Test, Installationsprüftest) installieren und ausführen.

Informationen zu diesem Vorgang

Die Informationen in diesem Abschnitt gelten für die Installation und den Test des Ressourcenadapters auf einem Server, der in einer verwalteten Domäne eingesetzt wird. Wird der Ressourcenadapter auf einem Standalone-Server installiert, finden Sie entsprechende Informationen unter „Installation und Test auf einem Standalone-Server” auf Seite 842.

Wichtig: Diese Anweisungen gelten nur für JBoss EAP 6.3 . Informationen zur Installation des Ressourcenadapters in JBoss AS 5.1 und 6 finden Sie in „Ressourcenadapter in JBoss AS 5.1 und 6 installieren und testen” auf Seite 839.

Vorgehensweise

1. Implementieren Sie den Ressourcenadapter auf Ihrem Server mithilfe der JBoss -Managementkonsole oder der Management-CLI.
2. Erstellen Sie die JMS-Ressourcen, die für die IVT-Anwendung erforderlich sind, indem Sie die folgenden Einträge zum Abschnitt <resource-adapters> der <EAP_HOME>/domain/configuration/domain.xml filehinzufügen:

```
<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.ManagedConnectionFactoryImpl"
         jndi-name="java:jboss/jms/ivt/IVTCF"
         enabled="true"
         use-java-context="true"
         pool-name="IVTCF">
      <config-property name="port">
          1414
      </config-property>
      <config-property name="hostName">
          localhost
      </config-property>
      <config-property name="channel">
          SYSTEM.DEF.SVRCONN
      </config-property>
      <config-property name="transportType">
          CLIENT
      </config-property>
      <config-property name="queueManager">
          ExampleQM
      </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>
```

3. Fügen Sie den Startparametern des Anwendungsservers die folgenden Informationen hinzu:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Stoppen Sie den Anwendungsserver und starten Sie ihn anschließend erneut.
5. Implementieren Sie die IVT-Anwendung mithilfe der JBoss -Managementkonsole oder der Management-CLI.
6. Führen Sie die IVT-Anwendung aus.

   Weitere Informationen finden Sie im Abschnitt „Installationsprüftestprogramm für den WebSphere MQ-Ressourcenadapter” auf Seite 832. Für JBosslautet die Standard-URL `http://localhost:8080/WMQ_IVT/`.

   Anmerkung: Die JNDI-Namen der für die Ausführung der IVT-Anwendung erforderlichen JMS-Ressourcen lauten wie folgt:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF
```

```
Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

Geben Sie beim Start der IVT-Anwendung über die oben angegebene URL die JNDI-Namen dieser Ressourcen in die entsprechenden Felder ein und klicken Sie auf **Run IVT** (Installationsprüftest ausführen).

In den WebSphere MQ-Klassen für JMS bereitgestellte Scripts

Es werden mehrere Scripts bereitgestellt, die Sie bei allgemeinen Tasks unterstützen, die bei der Verwendung von WebSphere MQ-Klassen für JMS ausgeführt werden müssen.

Tabelle 102 auf Seite 845 listet alle Scripts und ihre Verwendungen auf. Die Scripts befinden sich im Unterverzeichnis `bin` des Installationsverzeichnisses der WebSphere MQ-Klassen für JMS.

Tabelle 102. In den WebSphere MQ-Klassen für 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 „Der Punkt-zu-Punkt-Installationsprüftest für WebSphere MQ Classes for JMS” auf Seite 824).
JMSAdmin[1]	Führt das WebSphere MQ-JMS-Verwaltungstool aus (siehe Beschreibung unter „Verwaltungstool der IBM WebSphere MQ classes for JMS aufrufen” auf Seite 992).
JMSAdmin.config	Die Konfigurationsdatei für das WebSphere MQ-JMS-Verwaltungstool (siehe Beschreibung unter „JMS-Verwaltungstool konfigurieren” auf Seite 993).
PSIVTRun [1]	Führt das Programm für den Installationsprüftest von Publish/Subscribe aus (siehe Beschreibung unter „Der Publish/Subscribe-Installationsprüftest für WebSphere MQ Classes for JMS” auf Seite 828).
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 WebSphere MQ -Klassen für JMS in einer 32-Bit-JVM (Java Virtual Machine) auf UNIX and Linux -Systemen wie in „Umgebungsvariablen, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden” auf Seite 766beschrieben fest.
setjmsenv64	Legt die Umgebungsvariablen für die Ausführung einer Anwendung, die die WebSphere MQ-Klassen für JMS verwendet, in einer 64-Bit-Version der JVM unter UNIX and Linux fest (siehe „Umgebungsvariablen, die von den IBM WebSphere MQ-Klassen für JMS verwendet werden” auf Seite 766).

Tabelle 102. In den WebSphere MQ-Klassen für JMS bereitgestellte Scripts (Forts.)	
Versorgungsbetrieb	**Verwenden Sie**
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 WebSphere 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.

IBM WebSphere MQ classes for JMS enthält die folgenden OSGi-Bundles.

com.ibm.msg.client.osgi.jms<Versionsnummer>.jar
Die gemeinsame Codeschicht in IBM WebSphere MQ classes for JMS. Informationen zur Schichtarchitektur von WebSphere MQ Classes for JMS finden Sie im Abschnitt „Schichtarchitektur” auf Seite 849.

com.ibm.msg.client.osgi.jms.prereq_<Versionsnummer>.jar
Die vorausgesetzten JAR-Dateien (Java Archive) für die allgemeine Schicht.

com.ibm.msg.client.osgi.commonservices.j2se_<Versionsnummer>.jar
Allgemeine Services für Anwendungen der Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_<Versionsnummer>.jar
Nachrichten für die allgemeine Schicht.

com.ibm.msg.client.osgi.wmq_<Versionsnummer>.jar
Der IBM WebSphere MQ-Messaging-Provider in IBM WebSphere MQ classes for JMS. Informationen zur Schichtarchitektur von IBM WebSphere MQ classes for JMS finden Sie im Abschnitt „Schichtarchitektur” auf Seite 849.

com.ibm.msg.client.osgi.wmq.prereq_<Versionsnummer>.jar
Die vorausgesetzten JAR-Dateien für den IBM WebSphere MQ-Messaging-Provider.

com.ibm.msg.client.osgi.wmq.nls_<Versionsnummer>.jar
Nachrichten für den IBM WebSphere MQ-Messaging-Provider.

com.ibm.mq.osgi.directip_< Versionsnummer > .jar
Die JAR-Dateien, mit deren Hilfe der IBM WebSphere MQ-Messaging-Provider eine Echtzeitverbindung zu einem Broker erstellen kann.

Dabei ist < Versionsnummer > die Versionsnummer von WebSphere MQ , die installiert wurde.

Die Bundles werden im Unterverzeichnis `java/lib/OSGi` Ihrer WebSphere MQ -Installation oder im Ordner `java\lib\OSGi` unter Windows installiert.

Das Produktpaket com.ibm.mq.osgi.java < Versionsnummer > .jar, das auch im Unterverzeichnis `java/lib/OSGi` Ihrer WebSphere MQ -Installation oder im Ordner `java\lib\OSGi` unter Windows installiert wird, ist Teil der WebSphere MQ -Klassen für Java. Dieses Bundle darf nicht in eine OSGi-Laufzeitumgebung geladen werden, in der WebSphere MQ Classes for JMS geladen wurde.

Die OSGi-Bundles für WebSphere 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 WebSphere 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 IBM WebSphere MQ Classes for JMS-Bundles bekannt sein, die Benutzerbundles werden jedoch nicht von den IBM WebSphere MQ classes for JMS-Bundles erkannt. Deshalb kann das Klassenladeprogramm, das in einem Bundle für die IBM WebSphere MQ classes for JMS verwendet wird, keine Kanalexitklasse laden, die sich in einem Benutzerbundle befindet.

Weitere Informationen über OSGi finden Sie auf der Website der Open Service Gateway-Initiative.

Probleme in Zusammenhang mit den IBM WebSphere MQ-Klassen für JMS beheben

Sie können Probleme untersuchen, indem Sie die Installationsprüfprogramme ausführen sowie Trace- und Protokollfunktionen nutzen.

Wenn ein Programm nicht erfolgreich ausgeführt wird, führen Sie wie in den Abschnitten „Der Punkt-zu-Punkt-Installationsprüftest für WebSphere MQ Classes for JMS" auf Seite 824 und „Der Publish/Subscribe-Installationsprüftest für WebSphere MQ Classes for JMS" auf Seite 828 beschrieben eines der Installationsprüfprogramme aus und halten Sie sich an die Empfehlung in den Diagnosenachrichten.

Protokollierung und IBM WebSphere MQ classes for JMS

Die Protokollausgabe wird standardmäßig an die Datei `mqjms.log` gesendet. Sie können sie jedoch an eine bestimmte Datei oder an ein bestimmtes Verzeichnis umleiten.

Die Protokollierungseinrichtung der IBM WebSphere MQ classes for JMS wird bereitgestellt, um schwerwiegende Probleme zu melden. Dies bezieht sich insbesondere auf Probleme, die nicht auf Programmierfehler, sondern auf Konfigurationsfehler hindeuten. Die Protokollausgabe wird standardmäßig an die Datei `mqjms.log` im JVM-Arbeitsverzeichnis gesendet.

Durch die Festlegung der Eigenschaft com.ibm.msg.client.commonservices.log.outputName können Sie die Protokollausgabe an eine andere Datei umleiten. Für diese Eigenschaft sind folgende Werte möglich:

- Ein einzelner Pfadname.
- Eine durch Kommas getrennte Liste mit Pfadnamen (alle Daten werden in allen Dateien protokolliert).

Jeder Pfadname kann Folgendes sein:

- Absolut oder relativ.
- `stderr` oder `System.err` für die Darstellung des Standardfehler-Datenstroms.
- `sttdout` oder `System.out` für die Darstellung des Standardausgabe-Datenstroms.

Wenn der Wert der Eigenschaft ein Verzeichnis angibt, wird die Protokollausgabe in die Datei `mqjms.log` in dem betreffenden Verzeichnis geschrieben. Wenn der Wert der Eigenschaft eine bestimmte Datei angibt, wird die Protokollausgabe in diese Datei geschrieben.

Sie können diese Eigenschaft in der Konfigurationsdatei der IBM WebSphere MQ classes for JMS oder als Systemeigenschaft im **java**-Befehl festlegen. Im folgenden Beispiel wird die Eigenschaft als Systemeigenschaft festgelegt und gibt eine bestimmte Datei an:

```
java -Djava.library.path=library_path
     -Dcom.ibm.msg.client.commonservices.log.outputName=/mydir/mylog.txt
      MyAppClass
```

In diesem Befehl steht *Bibliothekspfad* für den Pfad des Verzeichnisses, das die Bibliotheken der IBM WebSphere MQ classes for JMS enthält (siehe „JNI-Bibliotheken (Java Native Interface) konfigurieren" auf Seite 768).

Sie können die Protokollausgabe inaktivieren, indem Sie die Eigenschaft com.ibm.msg.client.commonservices.log.status auf OFF setzen. Der Standardwert dieser Eigenschaft ist ON.

Die Werte `System.err` und `System.out` können festgelegt werden, damit die Protokollausgabe an die Datenströme `System.err` und `System.out` gesendet wird.

Einführung in WebSphere MQ Classes for JMS (für Programmierer)

WebSphere MQ Classes for JMS) ist der JMS-Provider, der mit WebSphere MQ bereitgestellt wird. WebSphere MQ Classes for JMS implementiert die Schnittstellen, die im Paket 'javax.jms' definiert sind, und stellt darüber hinaus zwei Gruppen von Erweiterungen für die JMS-API bereit. Anwendungen mit Java Platform, Standard Edition (Java SE) und Java Platform, Enterprise Edition (Java EE) können WebSphere MQ Classes for JMS verwenden.

In der JMS-Spezifikation wird eine Gruppe von Schnittstellen definiert, die Anwendungen zur Durchführung von Messaging-Operationen verwenden können. Die aktuellste Version der Spezifikation ist Version 1.1. Das Paket 'javax.jms' spezifiziert die Details der JMS-Schnittstellen, und ein JMS-Provider implementiert diese Schnittstellen für ein bestimmtes Messaging-Produkt. WebSphere MQ Classes for JMS ist ein JMS-Provider, der die JMS-Schnittstellen für WebSphere MQ implementiert.

Der Logikfluss innerhalb einer JMS-Anwendung beginnt mit den Objekten 'ConnectionFactory' und 'Destination'. Mit dem Objekt ConnectionFactory erstellt die Anwendung ein Connection-Objekt, das die aktive Verbindung von der Anwendung zum Messaging-Server darstellt. Mit dem Objekt Connection erstellt die Anwendung ein Session-Objekt, das ein einzelner Threaded-Kontext für die Produzierung und Konsumierung von Nachrichten ist. Aus diesem Session-Objekt erstellt die Anwendung dann mit einem Destination-Objekt ein MessageProducer-Objekt, mit dessen Hilfe sie Nachrichten an das angegebene Ziel senden kann. Das Ziel ist entweder eine Warteschlange oder ein Thema im Messaging-System, das im Destination-Objekt gekapselt ist. Aus dem Session- und dem Destination-Objekt kann die Anwendung auch ein MessageConsumer-Objekt erstellen, über das sie die an das angegebene Ziel gesendeten Nachrichten empfängt.

Die JMS-Spezifikation setzt voraus, dass es sich bei den Objekten 'ConnectionFactory' und 'Destination' um verwaltete Objekte handelt. Ein Administrator erstellt und verwaltet verwaltete Objekte in einem zentralen Repository, und eine JMS-Anwendung ruft diese Objekte über JNDI (Java Naming and Directory Interface) ab. Das Repository mit den verwalteten Objekten kann jedes Repository von einer einfachen Datei bis hin zu einem Lightweight Directory Access Protocol (LDAP)-Verzeichnis sein.

WebSphere MQ Classes for JMS unterstützt die Verwendung von verwalteten Objekten. Eine Anwendung kann alle Funktionen von WebSphere MQ verwenden, die über WebSphere MQ Classes for JMS zugänglich sind, ohne dass WebSphere MQ-spezifische Informationen in der Anwendung selbst fest codiert werden müssen. Dank dieses Konzepts bleibt die Anwendung bis zu einem gewissen Grad unabhängig von der zugrunde liegenden WebSphere MQ-Konfiguration. Zur Erreichung dieser Unabhängigkeit kann die Anwendung JNDI zum Abrufen von Verbindungsfactorys und Zielen verwenden, die als verwaltete Objekte gespeichert sind, und zur Ausführung von Messaging-Operationen ausschließlich die im Paket javax.jms definierten Schnittstellen verwenden. Ein Administrator kann verwalteten Objekte mithilfe des JMS-Verwaltungstools für WebSphere MQ oder mithilfe von IBM WebSphere MQ Explorer in einem zentralen Repository erstellen und verwalten. Allerdings stellt ein Anwendungsserver in der Regel ein eigenes Repository für verwaltete Objekte und eigene Tools zum Erstellen und Pflegen dieser Objekte bereit. Eine Java EE -Anwendung kann daher JNDI verwenden, um verwaltete Objekte entweder aus dem Repository des Anwendungsservers oder aus einem zentralen Repository abzurufen.

WebSphere MQ Classes for JMS stellt auch Erweiterungen für die JMS-API bereit. In Vorgängerreleases von WebSphere MQ Classes for JMS sind Erweiterungen enthalten, die in MQConnectionFactory-, MQQueue- und MQTopic-Objekten implementiert werden. Diese Objekte verfügen über Eigenschaften und Methoden, die für WebSphere MQ spezifisch sind. Bei den Objekten kann es sich um verwaltete Objekte handeln oder eine Anwendung kann die Objekte dynamisch während der Laufzeit erstellen. Da diese Erweiterungen in diesem Release von WebSphere MQ Classes for JMS verwaltet werden, können Sie alle Anwendungen, die diese Erweiterungen verwenden, weiterhin ohne Änderung nutzen. Diese Erweiterungen werden als *WebSphere MQ-JMS-Erweiterungen* bezeichnet. Beachten Sie, dass in dieser Dokumentation Objekte, die während der Laufzeit dynamisch von einer Anwendung erstellt werden, *nicht* als verwaltete Objekte gelten.

Zusätzlich zu den WebSphere MQ-JMS-Erweiterungen stellt dieses Release von WebSphere MQ Classes for JMS einen allgemeineren Satz Erweiterungen für die JMS-API bereit. Diese Erweiterungen sind unter dem Begriff '*IBM JMS-Erweiterungen*' bekannt und haben die folgende allgemeine Zielsetzung:

- Bereitstellung höherer Konsistenz bei den verschiedenen IBM JMS-Providern
- Vereinfachung des Schreibens einer Brückenanwendung zwischen zwei IBM Messaging-Systemen
- Vereinfachung des Portierens einer Anwendung zwischen zwei IBM JMS-Providern

Hauptaufgabe dieser Erweiterungen ist die dynamische Erstellung und Konfiguration von Verbindungsfactorys und Zielen während der Laufzeit. Jedoch bieten diese Erweiterungen auch Funktionen, die nicht direkt mit Messaging in Verbindung stehen, beispielsweise Funktionen für die Fehlerbestimmung.

Ein Verbindungsfactory-, Queue- oder Topic-Objekt, das mit der Schnittstelle javax.jms oder einer der beiden JMS-Erweiterungssätze erstellt wurde, kann mit jeder dieser APIs adressiert werden; d. h., sie können für jede dieser Schnittstellen umgesetzt werden. Um die Portierbarkeit Ihrer Anwendungen zu erleichtern, sollten Sie stets die allgemeinste API verwenden, die für Ihre Anforderungen gerade noch geeignet ist.

Sowohl Java SE-als auch Java EE -Anwendungen können WebSphere MQ -Klassen für JMS verwenden. Auf der Plattform Java EE unterstützt WebSphere MQ Classes for JMS zwei Arten der Kommunikation zwischen einer Komponente einer Anwendung und einem WebSphere MQ -Warteschlangenmanager:

Abgehende Kommunikation
Direkt in der JMS-API erstellt eine Anwendungskomponente eine Verbindung zu einem Warteschlangenmanager, um dann Nachrichten zu senden und zu empfangen.

Die Anwendungskomponente kann beispielsweise ein Anwendungsclient, ein Servlet, eine JavaServer Page (JSP), eine Enterprise JavaBean (EJB) oder eine Message-driven Bean (MDB) sein. Bei dieser Art der Kommunikation stellt der Anwendungsserver-Container nur Low-Level-Funktionen für Messaging-Operationen bereit, beispielsweise Verbindungspooling und Thread-Management.

Eingehende Kommunikation
Eine beim Ziel ankommende Nachricht wird einer Message-driven Bean (MDB) zugestellt, die die Nachricht dann verarbeitet.

Java EE -Anwendungen verwenden MDBs zur asynchronen Verarbeitung von Nachrichten. Eine MDB agiert als JMS-Nachrichtenlistener und wird durch eine onMessage()-Methode implementiert, die festlegt, wie Nachrichten verarbeitet werden. Ein MDB wird im EJB-Container des Anwendungsservers implementiert. Die Konfiguration eines MDB ist vom verwendeten Anwendungsserver abhängig. Jedoch muss in der Konfiguration angegeben sein, mit welchem Warteschlangenmanager die Verbindung hergestellt werden soll, wie diese Verbindung erfolgen soll, welches Ziel auf Nachrichten überwacht werden soll und wie sich das MDB hinsichtlich Transaktionen verhalten soll. Diese Informationen werden vom EJB-Container verwendet. Wenn eine Nachricht, die die Auswahlkriterien der MDB erfüllt, am angegebenen Ziel eintrifft, verwendet der EJB-Container WebSphere MQ Classes for JMS, um die Nachricht vom Warteschlangenmanager abzurufen, und übermittelt dann die Nachricht an die MDB, indem er seine onMessage()-Methode aufruft.

Architektur der IBM WebSphere MQ-Klassen für JMS

Die in IBM WebSphere MQ Version 7.0 bereitgestellten IBM WebSphere MQ-Klassen für JMS bieten im Vergleich zu den Vorgängerreleases einige funktionale Erweiterungen. Ein Teil dieser Erweiterungen resultiert aus Änderungen an der Implementierung der IBM WebSphere MQ-Klassen für JMS, während sich die anderen Erweiterungen aus der Tatsache ergeben, dass die IBM WebSphere MQ-Klassen für JMS Änderungen an der zugrunde liegenden IBM WebSphere MQ-Funktion nutzen.

In den folgenden Abschnitten werden die wichtigsten Erweiterungen zusammengefasst.

Schichtarchitektur

In früheren Releases von WebSphere MQ galt die Implementierung der WebSphere MQ-Klassen für JMS ausschließlich für WebSphere MQ. Andere IBM Produkte, die Messaging-Systeme verwenden, umfassen

ebenfalls JMS-Provider, diese haben jedoch nur sehr wenige oder keine Gemeinsamkeiten mit der Implementierung der WebSphere MQ-Klassen für JMS.

Ab WebSphere MQ V7.0 weisen WebSphere MQ-Klassen für JMS eine Schichtarchitektur auf. Die oberste Codeschicht ist eine allgemeine Schicht, die von jedem IBM JMS-Provider verwendet werden kann. Wenn eine Anwendung eine JMS-Methode aufruft, erfolgt die gesamte Verarbeitung des Aufrufs, der sich nicht speziell auf ein Messaging-System bezieht, durch die allgemeine Schicht, die auch eine konsistente Antwort auf den Aufruf bereitstellt. Die gesamte Verarbeitung des Aufrufs, die sich speziell auf ein Messaging-System bezieht, wird an eine untergeordnete Schicht delegiert. In Abbildung 125 auf Seite 850 ist die Schichtarchitektur dargestellt.

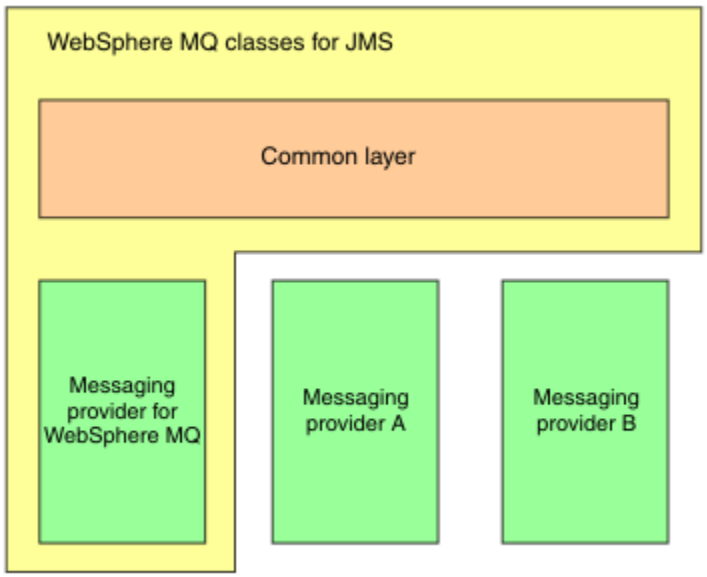


Abbildung 125. Schichtarchitektur für IBM JMS-Provider

Die Umstrukturierung in eine Schichtarchitektur verfolgt folgende Ziele:

- Verbesserung der Verhaltenskonsistenz verschiedener IBM JMS-Provider
- Vereinfachung des Schreiben seiner Brückenanwendung zwischen zwei IBM Messaging-Systemen
- Vereinfachung des Portierens einer Anwendung zwischen zwei IBM JMS-Providern

Diese Implementierung der WebSphere MQ-Klassen für JMS enthält jetzt außerdem eine Gruppe von Erweiterungen der JMS-API. Diese Erweiterungen werden als *IBM JMS-Erweiterungen* bezeichnet. Der Schwerpunkt dieser Erweiterungen liegt in der dynamischen Erstellung und Konfiguration von Verbindungsfactorys und Zielen während der Ausführung.

Eine Anwendung, die die IBM JMS-Erweiterungen nutzt, beginnt durch die Erstellung eines JmsFactory-Factory-Objekts, bei dem als Parameter eine Konstante angegeben wird, die das gewählte Messaging-System identifiziert. Die Anwendung verwendet das JmsFactoryFactory-Objekt zur Erstellung von Verbindungsfactorys und Zielen, die die richtigen spezialisierten Klassen für das gewählte Messaging-System aufweisen.

Die Anwendung kann dann die Verbindungsfactorys und Ziele durch die Festlegung der zugehörigen Eigenschaften konfigurieren. Die IBM JMS-Erweiterungen bieten eine Reihe von Methoden zur Festlegung von Eigenschaften. Diese Methoden agieren unabhängig von Messaging-Systemen. Jeder Datentyp verfügt über seine eigene Set-Methode und jede Eigenschaft wird über einen Namen identifiziert, der als statisches endgültiges Mitglied der WMQConstants-Klasse definiert ist. Wenn eine Anwendung eine dieser Methoden aufruft, ist einer der Parameter im Aufruf der Name der Eigenschaft, während der andere Parameter der Wert der Eigenschaft ist.

Handelt es sich bei dem Messaging-System beispielsweise um WebSphere MQ, ist eine der Eigenschaften einer Verbindungsfactory der Name des Warteschlangenmanagers, zu dem eine Verbindung hergestellt

werden soll. Mithilfe der IBM JMS-Erweiterungen legt eine Anwendung als Name des Warteschlangenmanagers JUPITER fest, indem sie folgende Methode aufruft:

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

Im Gegensatz hierzu kann eine Anwendung dieselbe Funktion auführen, indem Sie folgende Methode aufruft:

```
MQConnectionFactory myCF;
...
myCF.setQueueManager("JUPITER");
```

Diese Methode ist eine WebSphere MQ-JMS-Erweiterung, die speziell für das Messaging-System WebSphere MQ vorgesehen ist. Daher wird bei Verwendung dieser Methode die Portierung der Anwendung auf einen anderen IBM JMS-Provider möglicherweise erschwert.

Beziehung zwischen WebSphere MQ Classes for JMS und WebSphere MQ Classes for Java

In Releases von WebSphere MQvor Version 7.0wurde WebSphere MQ Classes for JMS fast vollständig als Codeebene über WebSphere MQ Classes for Java implementiert. Diese Anordnung hat bei Anwendungsentwicklern zu einigen Unklarheiten geführt, da die Festlegung von Feldern oder der Aufruf von Methoden in der MQEnvironment-Klasse unerwünschte und unerwartete Auswirkungen auf das Laufzeitverhalten des Codes haben kann, der unter Verwendung der WebSphere MQ-Klassen für JMS geschrieben wird. Darüber hinaus hatte die Implementierung von WebSphere MQ Classes for JMS einige Einschränkungen in Bereichen, in denen die JMS-API keine natürliche Ergänzung zu WebSphere MQ Classes for Java ist, und diese Einschränkungen haben zu einigen Problemen bei der Laufzeitleistung geführt.

Ab WebSphere MQ V7.0hängt die Implementierung von WebSphere MQ Classes for JMS nicht mehr von WebSphere MQ Classes for Java ab. WebSphere MQ Classes for Java und WebSphere MQ Classes for JMS sind jetzt Peers, die eine gemeinsame Java-Schnittstelle zur MQI verwenden. Diese Anordnung ermöglicht mehr Spielraum für die Leistungsoptimierung und bedeutet, dass die Festlegung von Feldern oder der Aufruf von Methoden in der MQEnvironment-Klasse keine Auswirkung auf das Laufzeitverhalten von Code hat, der unter Verwendung der WebSphere MQ-Klassen für JMS geschrieben wird.Abbildung 126 auf Seite 851 zeigt die Beziehung zwischen WebSphere MQ Classes for JMS und WebSphere MQ Classes for Java in früheren Releases von WebSphere MQ und in WebSphere MQ V7.0 und nachfolgenden Releases.

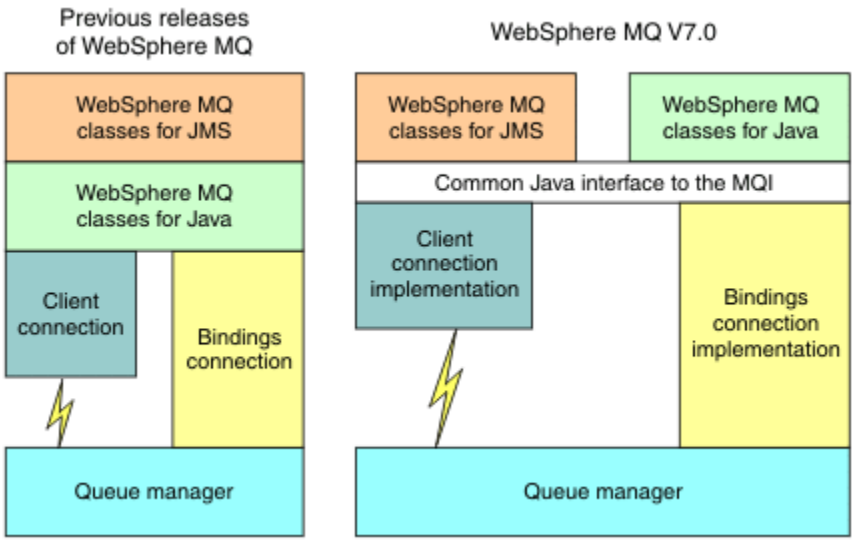


Abbildung 126. Beziehung zwischen WebSphere MQ Classes for JMS und WebSphere MQ Classes for Java

Um die Kompatibilität mit früheren Releases zu gewährleisten, können in Java geschriebene Kanalexitklassen weiterhin die WebSphere MQ -Klassen für Java-Schnittstellen verwenden, auch wenn die Kanalexitklassen von WebSphere MQ -Klassen für JMS aufgerufen werden. Die Verwendung der Schnittstellen von WebSphere MQ Classes for Java bedeutet jedoch, dass Ihre Anwendungen weiterhin von der JAR-Datei der WebSphere MQ Classes for Java, com.ibm.mq.jar, abhängig sind. Wenn Sie com.ibm.mq.jar nicht in Ihren Klassenpfad aufnehmen möchten, können Sie stattdessen die neue Schnittstellengruppe im Paket com.ibm.mq.exits verwenden.

JMS-verwaltete Objekte können jetzt mit dem WebSphere MQ-Explorer erstellt und konfiguriert werden.

Publish/Subscribe-Messaging

WebSphere MQ V7.0 und nachfolgende Releases enthalten eine eingebettete Publish/Subscribe-Funktion. Diese Funktion ersetzt die Komponente WebSphere MQ Publish/Subscribe, die im Lieferumfang von WebSphere MQ V6.0 enthalten war.

Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, können die eingebettete Publish/Subscribe-Funktion nutzen und diese anstelle von WebSphere Event Broker oder WebSphere Message Broker für das Publish/Subscribe-Messaging mit WebSphere MQ als Transportmethode verwenden. Die Konfiguration der WebSphere MQ-Klassen für JMS für die Verwendung der neuen Funktion ist einfacher als die Konfiguration der WebSphere MQ-Klassen für JMS für die Verwendung von WebSphere MQ Publish/Subscribe, WebSphere Event Broker oder WebSphere Message Broker. Administratoren und Anwendungsentwickler müssen Veröffentlichungswarteschlangen, Warteschlangen für Teilnehmerberechtigungen, Subskriptionsspeicher und Subskribentenbereinigungen nicht mehr verwalten. Außerdem weisen die ConnectionFactory- und Topic-Objekte weniger Eigenschaften auf.

Die eingebettete Publish/Subscribe-Funktion bietet darüber hinaus einige zusätzliche Features wie ständige Veröffentlichungen und die Auswahl zwischen zwei Platzhalterschemas für die Angabe einer Reihe von Themen, die eine Anwendung abonnieren möchte.

Eine Anwendung kann nach wie vor eine Echtzeitverbindung mit einem Broker von WebSphere Event Broker oder WebSphere Message Broker für das Publish/Subscribe-Messaging verwenden. Diese Unterstützung wurde nicht geändert.

Anwendungen, die WebSphere MQ Publish/Subscribe verwenden, können die eingebettete Publish/Subscribe-Funktion unverändert verwenden, wenn der Warteschlangenmanager, mit dem sie verbunden sind, aufgerüstet wird. Eigenschaften, die von einer Anwendung festgelegt werden, von der eingebetteten Publish/Subscribe-Funktion jedoch nicht benötigt werden, werden ignoriert.

WebSphere MQ-Messaging-Provider

Der Messaging-Provider von WebSphere MQ kann auf zwei Arten ausgeführt werden:

- *WebSphere MQ-Messaging-Provider im normalen Modus*
- *WebSphere MQ-Messaging-Provider im Migrationsmodus*

Im normalen Modus des WebSphere MQ-Messaging-Providers werden zur Implementierung von JMS alle Features der Warteschlangenmanager von WebSphere MQ Version 7.0 und der nachfolgenden Releases verwendet. Dieser Modus wird ausschließlich für die Verbindung mit einem WebSphere MQ-Warteschlangenmanager genutzt; eine Verbindung mit den Warteschlangenmanagern von WebSphere MQ Version 7.0 und nachfolgenden Releases ist sowohl im Client- als auch im Bindungsmodus möglich. Dieser Modus wurde für die Verwendung der neuen Funktion von WebSphere MQ Version 7.0 und nachfolgenden Releases optimiert.

Der Migrationsmodus des WebSphere MQ-Messaging-Providers basiert auf der Funktion von WebSphere MQ Version 6.0 und verwendet zur Implementierung des JMS nur die Features, die beim Warteschlangenmanager von WebSphere MQ Version 6.0 verfügbar waren. Es ist zwar möglich, im Migrationsmodus des WebSphere MQ-Messaging-Providers eine Verbindung zu Warteschlangenmanagern von WebSphere MQ Version 7.0 und nachfolgenden Releases herzustellen, die optimierten Funktionen der Version 7.0 können in diesem Fall jedoch nicht verwendet werden. In diesem Modus sind Verbindungen zu den beiden folgenden Warteschlangenmanagerversionen möglich:

1. WebSphere MQ-Warteschlangenmanager der Version 7.0 und nachfolgenden Releases im Bindungs- oder Clientmodus, wobei dieser Modus nur die Features nutzt, die einem Warteschlangenmanager von WebSphere MQ Version 6.0 zur Verfügung standen
2. WebSphere MQ-Warteschlangenmanager der Version 6.0 oder früher im Clientmodus

Wenn Sie mit WebSphere Event Broker oder WebSphere Message Broker unter Verwendung von WebSphere MQ Enterprise Transport eine Verbindung herstellen möchten, müssen Sie den Migrationsmodus des WebSphere MQ-Messaging-Providers verwenden. Beim Einsatz von WebSphere MQ Real-Time Transport wird der automatisch der Migrationsmodus des WebSphere MQ-Messaging-Providers ausgewählt, da Sie im Verbindungsfactoryobjekt explizit Eigenschaften ausgewählt haben. Bei der Herstellung einer Verbindung zu WebSphere Event Broker oder WebSphere Message Broker über WebSphere MQ Enterprise Transport gelten die allgemeinen Regeln für die Modusauswahl, die im Abschnitt Regeln für die Auswahl des Modus für den WebSphere MQ-Messaging-Provider beschrieben werden.

Asynchrone Nachrichtenverarbeitung

WebSphere MQ V7.0 und die nachfolgenden Releases unterstützen die asynchrone Nachrichtenverarbeitung. Eine Anwendung kann eine Rückruffunktion für ein Ziel registrieren. Wird eine geeignete Nachricht an das Ziel gesendet, ruft WebSphere MQ die Funktion auf und übergibt die Nachricht als Parameter. Die Funktion verarbeitet die Nachricht anschließend asynchron. In früheren Releases von WebSphere MQ war dieses Feature nur verfügbar, wenn die WebSphere MQ-Klassen für JMS verwendet wurden.

Die WebSphere MQ-Klassen für JMS wurde dahingehend geändert, dass jetzt dieses neue Feature in WebSphere MQ V7.0 und den nachfolgenden Releases genutzt wird. Die Implementierung von JMS-Nachrichtenlistenern ist jetzt besser in WebSphere MQ eingebunden und die WebSphere MQ-Klassen für JMS müssen kein Ziel mehr abfragen, um zu prüfen, ob eine geeignete Nachricht an das Ziel gesendet wurde. Folglich hat sich die Leistung der JMS-Nachrichtenlistener verbessert, insbesondere, wenn eine Anwendung mehrere Nachrichtenlistener in einer Sitzung zur Überwachung mehrerer Ziele verwendet. Der Nachrichtendurchsatz hat sich erhöht, und die Zustellung einer Nachricht an einen Nachrichtenlistener nach dem Eingang beim Ziel dauert nicht mehr so lange.

Nachrichtengesteuerte Beans (Message-driven Beans, MDBs) erzielen ähnliche Leistungssteigerungen. Aufgrund einer anderen Erweiterung der WebSphere MQ-Funktion entstehen bei mehreren nachrichtengesteuerten Beans, die Nachrichten desselben Ziels verarbeiten, jetzt weniger Konkurrenzsituationen im Hinblick auf die Nachrichten.

Nachrichtenauswahl

Mit Ausnahme der Nachrichtenauswahl über eine Nachrichten-ID oder Korrelations-ID erfolgte in Releases von WebSphere MQ vor Version 7.0 die gesamte Nachrichtenauswahl durch die WebSphere MQ-Klassen für JMS. In WebSphere MQ V7.0 und den nachfolgenden Releases erfolgt die gesamte Nachrichtenauswahl durch den Warteschlangenmanager.

Dies führt zu einem höheren Nachrichtendurchsatz bei Anwendungen, die Nachrichten über die Nachrichtenauswahl konsumieren. Die Leistungssteigerung ist bei einer Anwendung, die im Clientmodus eine Verbindung herstellt, höher, da nur die Nachrichten, die den Auswahlkriterien entsprechen, über das Netz transportiert werden und dieWebSphere MQ-Klassen für JMS nur die Nachrichten verarbeiten, die von ihm an die Anwendung zugestellt werden.

Kommunikationsverbindung gemeinsam nutzen

Wenn in früheren Releases von WebSphere MQ eine WebSphere MQ-Clientanwendung mehrfach über denselben MQI-Kanal eine Verbindung zu einem Warteschlangenmanager herstellte, erforderte jede Instanz des MQI-Kanals eine eigene TCP-Verbindung. In WebSphere MQ V7.0 und den nachfolgenden Releases kann jede Verbindung mit dem Warteschlangenmanager, bei der derselbe MQI-Kanal verwendet wird, eine einzelne TCP-Verbindung gemeinsam nutzen. Diese Anordnung bedeutet, dass weniger Netzressourcen erforderlich sind, und die Gesamtdauer, die für die Erstellung mehrerer Verbindungen zum Warteschlangenmanager benötigt wird, sich verringert. Dies gilt insbesondere bei der Verwendung von SSL, da der SSL-Handshake nur einmal beim Start der TCP-Verbindung stattfindet.

die WebSphere MQ-Klassen für JMS nutzen diese Erweiterung. Für eine Anwendung, die im Clientmodus eine Verbindung zu einem Warteschlangenmanager herstellt, erstellen die WebSphere MQ-Klassen für JMS möglicherweise mehr als eine Verbindung zu einem Warteschlangenmanager und verwendet dabei den MQI-Kanal mit dem Namen, der als Eigenschaft des ConnectionFactory-Objekts angegeben ist. All diese Verbindungen zum Warteschlangenmanager können jetzt eine einzelne TCP-Verbindung gemeinsam nutzen.

Vorauslesen bei Clientverbindungen

Wenn eine Anwendung eine Clientverbindung für die Verarbeitung nicht persistenter Nachrichten eines Ziels verwendet, kann das Ziel so konfiguriert werden, dass die WebSphere MQ-Klassen für JMS Nachrichten, die von Interesse sind, in einem Puffer speichern, bevor diese der Anwendung zugestellt werden. Diese Optimierung wird als *Vorauslesen* bezeichnet und kann von Anwendungen verwendet werden, die Nachrichten synchron konsumieren, indem die Methode receive() aufgerufen wird, und von Nachrichtenlistenern und nachrichtengesteuerten Beans, die Nachrichten asynchron konsumieren. Das Vorauslesen ist besonders bei Zielen effektiv, die eine hohe Anzahl an Nachrichten aufweisen, die schnell konsumiert werden müssen.

Bei persistenten Nachrichten wird das Vorauslesen nicht angewendet, da der Warteschlangenmanager die Nachrichten nach einem Fehler nicht mehr wiederherstellen könnte, wenn sie in einen Puffer eingelesen würden. Eine Anwendung, die Nachrichten aus einem Ziel konsumiert und dabei sowohl persistente als auch nicht persistente Nachrichten nutzt, kann das Vorausleseverfahren jedoch auf jeden Fall verwenden. Die Nachrichtenreihenfolge wird beibehalten, die Laufzeit profitiert nur im Fall nicht persistenter Nachrichten von den Vorteilen des Vorausleseverfahrens.

Wenn Sie vor der Entscheidung stehen, ob das Vorausleseverfahren genutzt werden soll, berücksichtigen Sie folgende Punkte:

- Wenn eine Anwendung Nachrichten aus einem Ziel konsumiert, das für das Vorausleseverfahren konfiguriert wurde, und die Anwendung aus irgendeinem Grund beendet wird, werden alle nicht persistenten Nachrichten, die derzeit im Puffer gespeichert sind, gelöscht.
- Wenn alle der folgenden Bedingungen zutreffen, werden Nachrichten, die an eine Warteschlange in einer Sitzung gesendet werden, möglicherweise nicht in der Reihenfolge empfangen, in der sie gesendet wurden:
  - Eine Anwendung verwendet in derselben Sitzung zwei Nachrichtenkonsumenten zum Verarbeiten der Nachrichten aus der Warteschlange.
  - Jeder Nachrichtenkonsument verwendet ein anderes Destination-Objekt für die Warteschlange.
  - Mindestens eines der Destination-Objekte ist für das Vorauslesen konfiguriert.

Nachrichten senden

Wenn eine Anwendung Nachrichten an ein Ziel sendet, kann das Ziel so konfiguriert werden, dass, wenn die Anwendung send() aufruft, die WebSphere MQ-Klassen für JMS die Nachricht an den Warteschlangenmanager weiterleiten und die Steuerung an die Anwendung zurückgeben, ohne zu ermitteln, ob der Warteschlangenmanager die Nachricht fehlerfrei erhalten hat. Die WebSphere MQ-Klassen für JMS können nur bei nicht persistenten Nachrichten und persistenten Nachrichten, die in einer ausgeführten Sitzung gesendet werden, auf diese Weise agieren.

Bei persistenten Nachrichten, die in einer Sitzung mit Transaktionsunterstützung gesendet werden, bestimmt die Anwendung schließlich, ob der Warteschlangenmanager die Nachrichten sicher empfangen hat, wenn er commit () aufruft. Für alle Nachrichten, die in einer Sitzung ohne Transaktionsunterstützung gesendet werden, gibt die Eigenschaft SENDCHECKCOUNT des Objekts ConnectionFactory an, wie viele Nachrichten gesendet werden sollen, bevor WebSphere MQ Classes for JMS überprüft, ob der Warteschlangenmanager die Nachrichten sicher empfangen hat.

Diese Optimierung ist vor allem für eine Anwendung von Vorteil, die im Clientmodus eine Verbindung zu einem Warteschlangenmanager herstellt und schnell nacheinander mehrere Nachrichten senden muss, jedoch nicht sofort für jede gesendete Nachricht eine Rückmeldung vom Warteschlangenmanager benötigt.

Kanalexits

Beim Aufruf durch dieWebSphere MQ-Klassen für JMS verhalten sich in C oder C++ geschriebene Kanalexitprogramme jetzt so, als ob sie von einem WebSphere MQ-Client aufgerufen wären. Die Leistung von in Java geschriebenen Kanalexitklassen wurde verbessert, und Sie können jetzt Kanalexitklassen unter Verwendung einer neuen Gruppe von Schnittstellen im Paket com.ibm.mq.exits schreiben, anstatt die Schnittstellen in WebSphere MQ Classes for Java zu verwenden.

Nachrichteneigenschaften

Eine JMS-Nachricht besteht aus mehreren Headerfeldern, einer Reihe von Eigenschaften und einem Hauptteil, der die Anwendungsdaten enthält. Eine WebSphere MQ-Nachricht besteht mindestens aus einem Nachrichtendeskriptor und den Anwendungsdaten.

Wenn eine Anwendung, dieWebSphere MQ-Klassen für JMS verwendet, eine JMS-Nachricht sendet, ordnen die WebSphere MQ-Klassen für JMS die JMS-Nachricht einer WebSphere MQ-Nachricht zu. Einige der JMS-Headerfelder und Eigenschaften werden Feldern im Nachrichtendeskriptor zugeordnet, während andere Feldern in einem zusätzlichen WebSphere MQ-Header zugeordnet werden, der als 'MQRFH2-Header' bezeichnet wird. Wenn eine Anwendung, die die WebSphere MQ-Klassen für JMS verwendet, eine JMS-Nachricht empfängt, wird von den WebSphere MQ-Klassen für JMS die regressive Zuordnung ausgeführt.

Eine Anwendung, die die Message Queue Interface (MQI) zum Empfang von Nachrichten einer Anwendung nutzt, die die WebSphere MQ-Klassen für JMS verwendet, muss daher einen MQRFH2-Header verarbeiten können. Kann die Anwendung einen MQRFH2-Header nicht verarbeiten, können die WebSphere MQ-Klassen für JMS über die TARGCLIENT-Eigenschaft des Destination-Objekts angewiesen werden, keinen MQRFH2-Header in die WebSphere MQ-Nachrichten aufzunehmen. Durch den Ausschluss des MQRFH2-Headers gehen aber die Informationen in einigen JMS-Headerfeldern und -Eigenschaften verloren.

Ähnlich gilt, dass eine Anwendung, die Nachrichten über die MQI an eine Anwendung sendet, die die WebSphere MQ-Klassen für JMS verwendet, in jede Nachricht einen MQRFH2-Header aufnehmen muss. Ist kein MQRFH2-Header enthalten, können die WebSphere MQ-Klassen für JMS nur die JMS-Headerfelder und Eigenschaften festlegen, die aus den Feldern in einem Nachrichtendeskriptor abgeleitet werden können.

WebSphere MQ V7.0 bietet zusätzliche Unterstützung für Anwendungen, die Nachrichten über die MQI aus Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden, empfangen bzw. an diese senden.

Wenn eine Anwendung über den Aufruf von MQGET eine Nachricht aus einer Anwendung empfängt, die die WebSphere MQ-Klassen für JMS verwendet, hat die Anwendung die Wahl zwischen folgenden Methoden zum Empfang der Nachricht:

1. Die Nachricht wird mit einem Nachrichtendeskriptor, einem MQRFH2-Header, der aus den JMS-Headerfeldern und Eigenschaften abgeleitete Daten enthält, und den Anwendungsdaten zugestellt.
2. Die Nachricht wird mit einem Nachrichtendeskriptor, den Anwendungsdaten und einer Reihe von Nachrichteneigenschaften zugestellt.

In der Option 2 stellt jede Nachrichteneigenschaft ein JMS-Headerfeld oder eine Eigenschaft dar, die ursprünglich von den WebSphere MQ-Klassen für JMS einem Feld in einem MQRFH2-Header zugeordnet wurde. Nach dem MQGET-Aufruf kann die Anwendung mit dem MQINQMP-Aufruf die Werte der Nachrichteneigenschaften abrufen. Die Verwendung der Option 2 anstelle der Option 1 für den Empfang einer Nachricht vereinfacht die Anwendungslogik auf folgende Arten:

- Die Anwendung muss den variablen Teil des MQRFH2-Headers, der das JMS-Headerfeld und Eigenschaftsdaten enthält, die in einem XML-ähnlichen Format codiert sind, nicht syntaktisch analysieren.
- Die Anwendung muss die Zeichendaten im variablen Teil des MQRFH2-Headers nicht konvertieren.

Entsprechend gilt, dass eine Anwendung vor dem Aufruf von MQPUT zum Senden einer Nachricht an eine Anwendung, die die WebSphere MQ-Klassen für JMS verwendet, mit dem Aufruf MQSETMP die Werte der Nachrichteneigenschaften festlegen kann, statt einen MQRFH2-Header zu erstellen.

Servicefreundlichkeit

Die WebSphere MQ-Klassen JMS enthalten eine Reihe von Verbesserungen im Hinblick auf die Servicefreundlichkeit:

- Tracing.

  Die WebSphere MQ-Klassen für JMS enthalten eine Klasse, die von einer Anwendung zur Steuerung der Tracefunktion genutzt werden kann. Eine Anwendung kann das Tracing starten und stoppen, die erforderliche Detaillierungsebene in einem Trace bestimmen und die Traceausgabe auf verschiedene Weise anpassen.

- Protokollierung.

  Die WebSphere MQ-Klassen für JMS pflegen eine Protokolldatei, die Nachrichten zu Fehlern enthält, welche behoben werden müssen. Die Nachrichten sind in Klartext verfasst. Die WebSphere MQ-Klassen für JMS enthalten eine Klasse, die von einer Anwendung für die Angabe der Protokolldateiposition und ihrer maximalen Größe verwendet werden kann.

- First Failure Support Technology (FFST).

  Wenn ein schwerwiegender Fehler auftritt, generieren die WebSphere MQ-Klassen für JMS einen FFST-Bericht in einer FDC-Datei. Der FFST-Bericht enthält Informationen, mit denen das Problem von IBM Service schneller diagnostiziert werden kann.

- Versionsinformationen.

  Die WebSphere MQ-Klassen für JMS enthalten eine Klasse, mit der eine Anwendung die Version der WebSphere MQ-Klassen für JMS abfragen kann.

- Ausnahmebedingungsnachrichten.

  Die Ausnahmebedingungsnachrichten wurden erweitert und bieten jetzt mehr Informationen zu den Fehlerursachen und den Maßnahmen, die zur Fehlerbehebung erforderlich sind.

- Anwendungsserver.

  Die Integration der Features der Servicefreundlichkeit der WebSphere MQ-Klassen für JMS in die Features von WebSphere Application Server wurden verbessert.

MQC wird durch MQConstants ersetzt

Zusammen mit IBM WebSphere MQ Version 7.0 wird das neue Paket com.ibm.mq.constants bereitgestellt. Dieses Paket enthält die Klasse 'MQConstants', mit der eine Reihe von Schnittstellen implementiert wird. 'MQConstants' enthält Definitionen aller Konstanten, die sich in der MQC-Schnittstelle befanden, sowie eine Reihe neuer Konstanten. Die Namen der Schnittstellen in diesem Paket sind eng an die Namen der in IBM WebSphere MQ verwendeten Headerdateien für Konstanten angelehnt.

Die Schnittstelle CMQC beispielsweise enthält die Konstante MQOO_INPUT_SHARED. Diese Schnittstelle und Konstante entsprechen der Headerdatei `cmqc.h` und der Konstanten MQOO_INPUT_SHARED.

com.ibm.mq.constants kann mit IBM WebSphere MQ -Klassen für Java und IBM WebSphere MQ -Klassen für JMS verwendet werden.

MQC ist nach wie vor vorhanden und enthält immer noch dieselben Konstanten. Für alle neuen Anwendungen muss allerdings das Paket "com.ibm.mq.constants" verwendet werden.

Anwendungen erstellen, die die WebSphere MQ-Klassen für JMS verwenden

Nach einer kurzen Einführung in das JMS-Modell finden Sie in diesem Abschnitt eine ausführliche Anleitung für das Schreiben von Anwendungen, die die WebSphere MQ-Klassen für JMS verwenden.

Das JMS-Modell

Das JMS-Modell definiert eine Gruppe von Schnittstellen, die Java-Anwendungen verwenden können, um Messaging-Operationen auszuführen. WebSphere MQ Classes for JMS als JMS-Provider definiert die

Beziehungen von JMS-Objekten zu WebSphere MQ-Konzepten. Die JMS-Spezifikation erwartet, dass es sich bei bestimmten JMS-Objekten um verwaltete Objekte handelt.

Die JMS-Spezifikation und das Paket javax.jms definieren eine Gruppe von Schnittstellen, die Java-Anwendungen verwenden können, um Messaging-Operationen auszuführen. In der folgenden Liste sind die wichtigsten JMS-Schnittstellen zusammengefasst:

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 127 auf Seite 857 sind diese Objekte und ihre Beziehungen dargestellt.

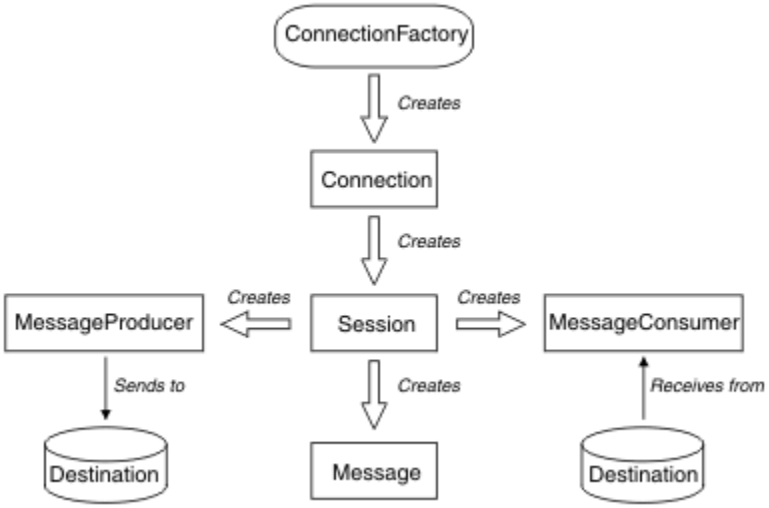


Abbildung 127. JMS-Objekte und ihre Beziehungen

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 Messaging-Stile:

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

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

Bei JMS-Versionen vor JMS 1.1 verwendet die Programmierung für die Punkt-zu-Punkt-Domäne eine bestimmte Gruppe von Schnittstellen und Methoden, während die Programmierung für die Publish/Subscribe-Domäne eine andere Gruppe verwendet. Auch wenn sich beide ähneln, sind sie völlig getrennt voneinander. Bei 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 103 auf Seite 858 werden die domänenunabhängigen JMS-Schnittstellen und ihre entsprechenden domänenspezifischen Schnittstellen aufgelistet.

Tabelle 103. 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
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

Da alle domänenspezifischen Schnittstellen in JMS 2.0 beibehalten wurden, können bestehende Anwendungen diese Schnittstellen nach wie vor verwenden. Für neue Anwendungen sollten Sie jedoch die domänenunabhängigen Schnittstellen in Erwägung ziehen.

In WebSphere MQ Classes for JMS bestehen zwischen JMS-Objekten und WebSphere MQ-Konzepten die folgenden Beziehungen:

- 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 bindet eine WebSphere MQ-Verbindungskennung ein, die daher den Transaktionsumfang der Sitzung definiert.
- Ein MessageProducer-Objekt und ein MessageConsumer-Objekt binden eine WebSphere MQ-Objektkennung ein.

Bei der Verwendung von WebSphere MQ Classes for JMS gelten alle normalen Regeln von WebSphere 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 setzt voraus, dass es sich bei den Objekten 'ConnectionFactory' und 'Destination' um verwaltete Objekte handelt. Ein Administrator erstellt und verwaltet verwaltete Objekte in einem zentralen Repository, und eine JMS-Anwendung ruft diese Objekte über JNDI (Java Naming and Directory Interface) ab.

In WebSphere MQ Classes for JMS ist die Implementierung der Destination-Schnittstelle eine abstrakte Oberklasse von Queue und Topic. Daher ist eine Instanz von Destination entweder ein Queue-Objekt oder ein Topic-Objekt. 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 WebSphere MQ Classes for JMS können Objekte der folgenden Typen also verwaltete Objekte sein:

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

JMS-Nachrichten

JMS-Nachrichten setzen sich aus einem Header, Eigenschaften und einem Hauptteil zusammen. JMS definiert fünf Arten eines Nachrichtenhauptteils.

JMS-Nachrichten setzen sich aus folgenden Teilen zusammen:

Kopfzeile
Alle Nachrichten unterstützen dieselbe Gruppe von Headerfeldern. Headerfelder enthalten Werte, die sowohl von Clients als auch Providern zur Identifizierung und Weiterleitung von Nachrichten genutzt werden.

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

Hauptteil
JMS definiert mehrere Arten des Nachrichtenhauptteils, die die Mehrheit der derzeit verwendeten Nachrichtendarstellungen abdecken.

JMS definiert fünf Arten eines Nachrichtenhauptteils:

Datenstrom
Ein Datenstrom mit primitiven Java-Werten Er wird nacheinander gefüllt und gelesen.

Zuordnung
Eine Gruppe von Name/Wert-Paaren, bei denen Namen Zeichenfolgen und Werte primitive Java-Typen sind. Die Einträge können nacheinander oder nach dem Namen aufgerufen werden. Die Reihenfolge der Einträge ist nicht definiert.

Text
Eine Nachricht, die eine 'java.lang.String' enthält.

Objekt
Eine Nachricht, die ein serialisierbares Java-Objekt enthält

Bytes
Ein Datenstrom nicht interpretierter Bytes. Dieser Nachrichtentyp ist für die buchstäbliche Codierung eines Hauptteils vorgesehen, damit dieser einem bestehenden Nachrichtenformat entspricht.

Das Headerfeld 'JMSCorrelationID' wird zum Verknüpfen einer Nachricht mit einer anderen verwendet. Für gewöhnlich verknüpft es eine Antwortnachricht mit der zugehörigen Anforderungsnachricht. Das Feld 'JMSCorrelationID' kann eine providerspezifische Nachrichten-ID, eine anwendungsspezifische Zeichenfolge oder einen providernativen Byte[]-Wert enthalten.

Nachrichtenselektoren in JMS

Nachrichten können anwendungsdefinierte Eigenschaftswerte enthalten. Eine Anwendung kann Nachrichtenselektoren verwenden, damit ein JMS-Provider Nachrichten filtert.

Eine Nachricht enthält eine integrierte Funktion zur Unterstützung anwendungsdefinierter Eigenschaftswerte. Tatsächlich wird damit ein Mechanismus bereitgestellt, mit dem anwendungsspezifische Headerfelder einer Nachricht hinzugefügt werden können. Eigenschaften ermöglichen es einer Anwendung, unter Verwendung von Nachrichtenselektoren einen JMS-Provider zu veranlassen, für sie Nachrichten mithilfe von anwendungsspezifischen Kriterien auszuwählen oder zu filtern. Anwendungsdefinierte Eigenschaften müssen die folgenden Regeln einhalten:

- Eigenschaftsnamen müssen die Regeln für eine Nachrichtenselektor-ID einhalten.
- Für Eigenschaftswerte sind folgende Werte möglich: Boolean, byte, short, int, long, float, double und String.
- Die Präfixe JMSX und JMS_, gefolgt vom Namen, sind reserviert.

Eigenschaftswerte werden vor dem Senden einer Nachricht festgelegt. Wenn ein Client eine Nachricht erhält, sind die Nachrichteneigenschaften schreibgeschützt. Wenn ein Client versucht, zu diesem Zeitpunkt Eigenschaften festzulegen, wird eine Ausnahmebedingung (MessageNotWriteableException) ausgelöst. Wenn 'clearProperties' aufgerufen wird, können die Eigenschaften gelesen und Werte in sie geschrieben werden.

Ein Eigenschaftswert kann unter Umständen einen Wert in einem Nachrichtenhauptteil duplizieren. JMS definiert keine Richtlinie dafür, was als Eigenschaft festgelegt werden kann. Anwendungsentwickler müssen jedoch beachten, dass JMS-Provider Daten in einem Nachrichtenhauptteil wahrscheinlich effizienter verarbeiten als Daten in Nachrichteneigenschaften. Um eine optimale Leistung zu erzielen, dürfen Anwendungen Nachrichteneigenschaften nur verwenden, wenn sie einen Nachrichtenheader anpassen müssen. Der Hauptgrund für diese Aktion ist die Unterstützung einer angepassten Nachrichtenauswahl.

Ein JMS-Nachrichtenselektor ermöglicht es einem Client, unter Verwendung des Nachrichtenheaders die Nachrichten anzugeben, an denen er interessiert ist. Es werden nur Nachrichten zugestellt, die mit dem Selektor übereinstimmen.

Nachrichtenselektoren können keine Nachrichtenhauptteilwerte referenzieren.

Ein Nachrichtenselektor ergibt eine Übereinstimmung mit einer Nachricht, wenn der Selektor beim Ersetzen des Nachrichtenheaderfelds und der Eigenschaftswerte durch ihre entsprechenden IDs im Selektor in 'wahr' ausgewertet wird.

Ein Nachrichtenselektor ist eine Zeichenfolge (String), deren Syntax auf einer Untergruppe der SQL92-Syntax für Bedingungsausdrücke basiert. Die Reihenfolge der Auswertung eines Nachrichtenselektors erfolgt innerhalb einer Rangfolgestufe von links nach rechts. Sie können diese Reihenfolge durch die Verwendung von Klammern ändern. Vordefinierte Selektorliterale und Operatorbezeichnungen werden hier in Großbuchstaben geschrieben, die Groß-/Kleinschreibung muss jedoch nicht beachtet werden.

Ein Selektor kann Folgendes enthalten:

- Literale
  - Ein Zeichenfolgeliteral wird in Anführungszeichen eingeschlossen. Ein doppeltes Anführungszeichen steht für ein Hochkomma. Beispiele: 'literal' und 'literal''s'. Wie Java-Zeichenfolgeliterale verwenden diese die Unicode-Zeichencodierung.
  - Ein exaktes numerisches Literal ist ein numerischer Wert ohne Dezimalzeichen, zum Beispiel 57, -957 und +62. Zahlen im Bereich von Java Long werden unterstützt.
  - Ein näherungsweise berechnetes numerisches Literal ist ein numerischer Wert in Exponentialschreibweise (zum Beispiel 7E3 oder -57.9E2) oder ein numerischer Wert mit einer Dezimalstelle (zum Beispiel 7,, -95,7 oder +6,2). Zahlen im Bereich von Java Double werden unterstützt.

  - TRUE und FALSE sind boolesche Literale.
- Kennungen:
  - Eine Kennung ist eine Folge von Java-Buchstaben und Java-Ziffern mit unbegrenzter Länge, wobei der erste Buchstabe ein Java-Buchstabe sein muss. Ein Buchstabe ist ein beliebiges Zeichen, für das die Methode 'Character.isJavaLetter' den Wert 'true' zurückgibt. Dies schließt _ und $ein. 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 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.
- Leerzeichen sind dasselbe wie für Java: Leerzeichen, Horizontaltabulator, Formularvorschub 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-Umstufung 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 numerische Java-Umstufung verwenden.
- 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 entspricht 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.

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"
```

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.

SQL unterstützt Vergleich und Arithmetik fester Dezimalzahlen, JMS-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.

Zuordnung von JMS-Nachrichten zu WebSphere MQ-Nachrichten

WebSphere MQ-Nachrichten setzen sich aus einem Nachrichtendeskriptor, einem optionalen MQRFH2-Header und einem Hauptteil zusammen. Der Inhalt einer JMS-Nachricht wird einer WebSphere MQ-Nachricht teilweise zugeordnet und teilweise in diese kopiert.

In diesem Abschnitt wird beschrieben, wie die im ersten Teil dieses Abschnitts beschriebene JMS-Nachricht einer WebSphere MQ-Nachricht zugeordnet wird. Diese Informationen sind vor allem für Programmierer bestimmt, die Nachrichten zwischen JMS-Anwendungen und herkömmlichen WebSphere MQ-Anwendungen übertragen möchten. Darüber hinaus sind diese Informationen für Personen interessant, die Nachrichten bearbeiten möchten, welche zwischen zwei JMS-Anwendungen übertragen wurden; dies kann beispielsweise in einer WebSphere Message Broker-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 WebSphere MQ-Warteschlangen oder -Nachrichten beteiligt.

WebSphere MQ-Nachrichten bestehen aus drei Komponenten:

- dem WebSphere MQ Message Descriptor (MQMD)
- einem WebSphere MQ-MQRFH2-Header
- Nachrichtenhauptteil

Der MQRFH2-Header ist optional und seine Aufnahme in eine abgehende Nachricht wird über ein Attribut in der JMS-Klasse Destination gesteuert. Sie können dieses Attribut mithilfe des JMS-Verwaltungstools von WebSphere MQ festlegen. Da der MQRFH2-Header JMS-spezifische Informationen enthält, muss er immer in die Nachricht eingeschlossen werden, wenn dem Absender bekannt ist, dass das Empfangsziel eine JMS-Anwendung ist. Wird eine Nachricht direkt an eine JMS-fremde Anwendung gesendet, wird der MQRFH2-Header normalerweise nicht eingeschlossen. Dies ist darauf zurückzuführen, dass eine derartige Anwendung keinen MQRFH2-Header in ihrer WebSphere 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.

Abbildung 128 auf Seite 864 zeigt, wie die Struktur einer JMS-Nachricht in eine WebSphere MQ-Nachricht umgewandelt und wieder zurückgewandelt wird:

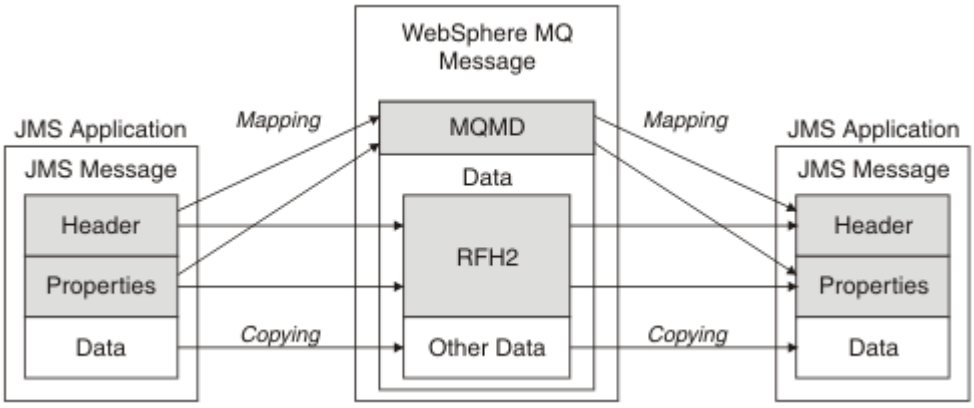


Abbildung 128. Vorgehensweise bei der Transformation von Nachrichten zwischen JMS und WebSphere MQ unter Verwendung des MQRFH2-Headers

Die Strukturen werden auf zwei Arten transformiert:

Zuordnen
Wenn der MQMD ein Feld enthält, das dem JMS-Feld entspricht, wird das JMS-Feld dem MQMD-Feld zugeordnet. Zusätzliche MQMD-Felder werden als JMS-Eigenschaften verfügbar gemacht, da eine JMS-Anwendung diese Felder bei der Kommunikation mit einer JMS-fremden Anwendung möglicherweise abruft oder festlegt.

Kopieren
Wenn keine MQMD-Entsprechung vorhanden ist, wird ein JMS-Headerfeld oder eine entsprechende Eigenschaft, die möglicherweise transformiert wurde, als Feld innerhalb des MQRFH2-Headers übergeben.

Der MQRFH2-Header und JMS
In dieser Themensammlung wird der MQRFH Version 2-Header beschrieben, der JMS-spezifische Daten enthält, die dem Nachrichteninhalt zugeordnet werden. Der MQRFH2 Version 2-Header ist erweiterbar und kann daher auch zusätzliche Informationen enthalten, die JMS nicht direkt zugeordnet sind. In diesem Abschnitt wird jedoch nur die Verwendung durch JMS beschrieben.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 WebSphere MQ-Headers modelliert und besteht aus folgenden Feldern:

StrucId (MQCHAR4)
Struktur-ID.

Muss MQRFH_STRUC_ID sein (Wert: "RFH") (Anfangswert).

MQRFH_STRUC_ID_ARRAY (Wert: "R", "F", "H", " ") wird ebenfalls definiert.

Version (MQLONG)
Strukturversionsnummer.

Muss MQRFH_VERSION_2 sein (Wert: 2) (Anfangswert).

StrucLength (MQLONG)
Gesamtlänge des MQRFH2, einschließlich der NameValueData-Felder.

Der Wert, der in 'StrucLength' festgelegt ist, muss ein Vielfaches von 4 sein (die Daten in den NameValueData-Feldern können mit Leerzeichen aufgefüllt werden, um dies zu erreichen).

Encoding (MQLONG)
Datencodierung.

Codierung von numerischen Daten im Nachrichtenabschnitt, der dem MQRFH2 folgt (der nächste Header oder die Nachrichtendaten, die diesem Header folgen).

CodedCharSetId (MQLONG)
Die ID des codierten Zeichensatzes.

Darstellung von beliebigen Zeichendaten im Nachrichtenabschnitt, der dem MQRFH2 folgt (der nächste Header oder die Nachrichtendaten, die diesem Header folgen).

Format (MQCHAR8)
Formatname.

Formatname für den Nachrichtenabschnitt, der dem MQRFH2 folgt.

Flags (MQLONG)
Flags.

MQRFH_NO_FLAGS =0. Es sind keine Flags festgelegt.

NameValueCCSID (MQLONG)
Die ID des codierten Zeichensatzes (CCSID) für die NameValueData-Zeichenfolgen, die in diesem Header enthalten sind. Die NameValueData-Zeichenfolgen können in einem Zeichensatz codiert werden, der sich von den anderen Zeichenfolgen unterscheidet, die im Header enthalten sind (StrucID und Format).

Wenn es sich bei der NameValueCCSID um eine 2-Byte-Unicode-CCSID (1200, 13488 oder 17584) handelt, ist die Byteanordnung des Unicode dieselbe wie die Byteanordnung der numerischen Felder im MQRFH2. (Beispiel: Version, StrucLength und NameValueCCSID selbst.)

Tabelle 104. Mögliche Werte für das NameValueCCSID-Feld

Wertschöpfung	Bedeutet
1200	UCS2 (mit offenem Ende)
1208	UTF8
13488	UCS2 2.0 Subset
17584	UCS2 2.1 Subset (einschließlich Euro-Symbol)

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 jeden der folgenden Ordner enthalten:

Ordner <mcd>

`mcd` enthält Eigenschaften, die das Format der Nachricht beschreiben. Beispielsweise identifiziert die Eigenschaft `Msd` (Message Service Domäne = Nachrichtenservicedomäne) eine JMS-Nachricht als `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` oder 0.

Der Ordner `mcd` ist immer in einer JMS-Nachricht mit `MQRFH2` enthalten.

Er ist immer in einer Nachricht mit einem `MQRFH2`-Header enthalten, die vom WebSphere Message Broker gesendet wurde. Er beschreibt die Domäne, das Format, den Typ und die Nachrichtengruppe einer Nachricht.

Tabelle 105. mcd-Eigenschaftsname, -Synonym, -Datentyp und -Ordner			
Eigenschaftssynonym	**Eigenschaftsname**	**Datenart**	**Ordner**
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

Fügen Sie keine eigenen Eigenschaften zum Ordner mcd hinzu.

Ordner <jms>

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

Ordner <usr>

usr enthält anwendungsdefinierte JMS-Eigenschaften, die der Nachricht zugeordnet sind. Der Ordner usr ist nur dann vorhanden, wenn eine Anwendung eine anwendungsdefinierte Eigenschaft festgelegt hat.

Ordner <mqext>

mqext enthält Eigenschaften, die ausschließlich von WebSphere Application Server verwendet werden. Der Ordner ist nur vorhanden, wenn die Anwendung mindestens eine der von IBM definierten Eigenschaften festgelegt hat.

Tabelle 106. mqext-Eigenschaftsname, -Synonym, -Datentyp und -Ordner			
Eigenschaftssynonym	**Eigenschaftsname**	**Datenart**	**Ordner**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>

Fügen Sie keine eigenen Eigenschaften zum Ordner mqext hinzu.

Ordner <mqps>

mqps enthält Eigenschaften, die nur von IBM WebSphere MQ Publish/Subscribe verwendet werden. Der Ordner ist nur vorhanden, wenn die Anwendung mindestens eine der integrierten Publish/Subscribe-Eigenschaften festgelegt hat.

Tabelle 107. mqps-Eigenschaftsname, -Synonym, -Datentyp und -Ordner			
Eigenschaftssynonym	**Eigenschaftsname**	**Datenart**	**Ordner**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>

Tabelle 107. mqps-Eigenschaftsname, -Synonym, -Datentyp und -Ordner (Forts.)			
Eigenschafts-synonym	**Eigen-schaftsna-me**	**Daten-art**	**Ordner**
MQSubUser-Data	mqps.Sud	strin g	<mqps><Sud>*subscriberUserData*</Sud></mqps>
MQIsRetai-ned	mqps.Ret	boo-lean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOpti-ons	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	strin g	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeq-Num	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></ mqps>
MQPubSt-rIntData	mqpse.Sid	strin g	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFor-mat	mqpse.Pfm t	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Fügen Sie keine eigenen Eigenschaften zum Ordner mqps hinzu.

In Tabelle 108 auf Seite 867 ist eine vollständige Liste der Eigenschaftsnamen aufgeführt.

Tabelle 108. Von JMS verwendete MQRFH2-Ordner und -Eigenschaften				
JMS-Feldname	**Java-Typ**	**MQRFH2-Ord-nername**	**Eigenschaftsna-me**	**Typ/Werte**
JMSDestination	Ziel	jms	Dst	Zeichenfolge
JMSExpiration	lang	jms	Erw	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Zeichenfolge	jms	Cid	Zeichenfolge
JMSReplyTo	Ziel	jms	Rto	Zeichenfolge
JMSTimestamp	lang	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 abschließenden Tag `</foldername>` können Leerzeichen als Füllzeichen folgen. Jedes Triplet wird mithilfe einer XML-ähnlichen Syntax wie folgt codiert:

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

Das Element `dt='datatype'` ist optional und wird für viele Eigenschaften weggelassen, da der Datentyp vordefiniert ist. Wenn sie enthalten ist, muss mindestens ein Leerzeichen vor dem Tag `dt=` stehen.

name
Der Name der Eigenschaft; siehe Tabelle 108 auf Seite 867.

datatype
Muss nach dem Umsetzen einem der in Tabelle 109 auf Seite 868 aufgeführten Datentypen entsprechen.

value
Eine Zeichenfolgedarstellung des zu übertragenden Werts, wobei die Definitionen in Tabelle 109 auf Seite 868 verwendet werden.

Ein Nullwert wird mit folgender Syntax codiert:

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

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

Tabelle 109. Eigenschaftsdatentypen

Datenart	Definition
Zeichenfolge	Beliebige Zeichenfolge mit Ausnahme von < und &
boolean	Das Zeichen `0` oder `1` (`0` = false, `1` = true)
bin.hex	Durch Hexadezimalziffern dargestellte Oktette
i1	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -128 bis einschließlich 127 liegen.
i2	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -32768 bis einschließlich 32767 liegen.
i4	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -2147483648 bis einschließlich 2147483647 liegen.
i8	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im Bereich -9223372036854775808 bis einschließlich 9223372036854775807 liegen.

Tabelle 109. Eigenschaftsdatentypen (Forts.)	
Datenart	**Definition**
int	Eine Zahl, ausgedrückt durch Ziffern `0..9`, mit optionalem Vorzeichen (keine Brüche oder Exponenten). Muss im selben Bereich wie i8 liegen. Dies kann anstelle eines i*-Typen verwendet werden, wenn der Sender der Eigenschaft keine spezielle Genauigkeit zuordnen will.
r4	Gleitkommazahl, Ausmaß < = 3.40282347E+38, > = 1.175E-37 ausgedrückt durch Ziffern `0..9`, optionales Vorzeichen, optionale Nachkommastellen, optionaler Exponent
r8	Gleitkommazahl, Größe < = 1.7976931348623E+308, > = 2.225E-307 ausgedrückt durch Ziffern `0..9`, optionales Vorzeichen, optionale Nachkommastellen, optionaler Exponent

Ein Zeichenfolgewert kann Leerzeichen enthalten. Sie müssen die folgenden Escapezeichenfolgen in einem Zeichenfolgewert verwenden:

- `&` für das Zeichen &
- < für das Zeichen <

Sie können die folgenden Escapezeichenfolgen verwenden, sie sind jedoch nicht erforderlich:

- `>` für das Zeichen >
- `&apos;` für das Zeichen '
- `&quot;` für das Zeichen "

JMS-Felder und -Eigenschaften mit entsprechenden MQMD-Feldern
In diesen Tabellen werden die MQMD-Felder aufgelistet, die äquivalent zu JMS-Headerfeldern, JMS-Eigenschaften und JMS-Provider-spezifischen Eigenschaften sind.

In Tabelle 110 auf Seite 869 werden die JMS-Headerfelder aufgelistet, während in Tabelle 111 auf Seite 870 die JMS-Eigenschaften aufgeführt sind, die den MQMD-Feldern direkt zugeordnet werden.Tabelle 112 auf Seite 870 In sind die providerspezifischen Eigenschaften und die MQMD-Felder aufgelistet, denen sie zugeordnet sind.

Tabelle 110. 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 111. 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 112. 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
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 871	MQCHAR8
JMS_IBM_PutApplType	Int	PutApplType	MQLONG
JMS_IBM_Encoding	Int	Encoding	MQLONG
JMS_IBM_Character_Set	Zeichenfolge	CodedCharacterSetId„2” auf Seite 871	MQLONG
JMS_IBM_PutDate	Zeichenfolge	PutDate	MQCHAR8
JMS_IBM_PutTime	Zeichenfolge	PutTime	MQCHAR8

Tabelle 112. Zuordnung der JMS-Provider-spezifischen Eigenschaften zu MQMD-Feldern (Forts.)			
JMS-Provider-spezifische Eigenschaft	**Java-Typ**	**MQMD-Feld**	**C-Typ**
JMS_IBM_Last_Msg_In_Group	boole-an	MsgFlags	MQLONG

Anmerkung:

1. JMS_IBM_Format ist das Format des Nachrichtenhauptteils. Es kann durch die Anwendungseinstellung der Eigenschaft JMS_IBM_Format der Nachricht (beachten Sie die Begrenzung auf 8 Zeichen) definiert werden oder standardmäßig das WebSphere MQ-Format des Nachrichtenhauptteils je nach JMS-Nachrichtentyp als Wert annehmen. '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 Eigenschaftswert von JMS_IBM_Character_Set ist ein Zeichenfolgewert (String), der den Java-Zeichensatz enthält, der dem numerischen CodedCharacterSetId-Wert entspricht. Das MQMD-Feld CodedCharacterSetId enthält einen numerischen Wert, der der durch die Eigenschaft JMS_IBM_Character_Set festgelegten Zeichenfolge für den Java-Zeichensatz entspricht.

Zuordnung von JMS-Feldern zu WebSphere MQ-Feldern (abgehende Nachrichten)
Diese Tabellen zeigen, wie JMS-Header- und Eigenschaftsfelder bei der Ausführung von send()- oder publish()-Operationen MQMD- und MQRFH2-Feldern zugeordnet werden.

Tabelle 113 auf Seite 871 zeigt die Zuordnung der JMS-Headerfelder zu MQMD/RFH2-Feldern bei der Ausführung von send() oder publish().Tabelle 114 auf Seite 872 zeigt, wie JMS-Eigenschaften MQMD/RFH2 -Feldern beim Senden (send ()) oder Veröffentlichen (publish ()) zugeordnet werden. Tabelle 115 auf Seite 872 zeigt, wie JMS-Provider-spezifische Eigenschaften MQMD-Feldern beim Senden (send ()) oder Veröffentlichen (publish ()) zugeordnet werden,

Bei Feldern mit dem Hinweis "Definiert durch Nachrichtenobjekt" wird der Wert der JMS-Nachricht übertragen, den das Feld kurz vor Beginn der send()- oder publish()-Operation hatte. Der Wert in der JMS-Nachricht bleibt während der Operation unverändert.

Bei Feldern mit dem Hinweis "Definiert durch Sendemethode" wird bei Ausführung der send()- oder publish()-Operation ein Wert zugewiesen (der in der JMS-Nachricht enthaltene Wert wird ignoriert). Der Wert in der JMS-Nachricht wird entsprechend an den verwendeten Wert angepasst.

Felder mit dem Hinweis 'Nur Empfang' werden nicht übertragen und bleiben bei send()- oder publish()-Operationen in der Nachricht unverändert.

Tabelle 113. 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	Nachrichtenob-jekt
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Nachrichtenob-jekt

Tabelle 113. Zuordnung der Felder abgehender Nachrichten (Forts.)

Name des JMS-Headerfelds	MQMD-Feld für Übertragung	Kopfzeile	Festgelegt durch
JMSType		MQRFH2	Nachrichtenobjekt
JMSRedelivered			Nur Empfang

Anmerkung:

1. Das MQMD-Feld CodedCharacterSetId enthält einen numerischen Wert, der der durch die Eigenschaft JMS_IBM_Character_Set festgelegten Zeichenfolge für den Java-Zeichensatz entspricht.

Tabelle 114. Zuordnung der JMS-Eigenschaften abgehender Nachrichten

Name der JMS-Eigenschaft	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 115. 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
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

Tabelle 115. Zuordnung der JMS-Provider-spezifischen Eigenschaften abgehender Nachrichten (Forts.)			
Name der JMS-Provider-spezifischen Eigenschaft	**MQMD-Feld für Übertragung**	**Kopfzeile**	**Festgelegt durch**
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-Header-Felder 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 Destination-Objekts serialisiert, damit ein empfangender JMS ein funktional entsprechendes Destination-Objekt wiederherstellen kann. Das Feld MQRFH2 wird als URI codiert (der Abschnitt „Uniform Resource Identifiers (URIs)" auf Seite 935 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 935 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:

- JMS-Wert PERSISTENT entspricht MQPER_PERSISTENT
- 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
WebSphere 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. Für die Nachrichten-ID (messageID) in WebSphere MQ wird ein Binärwert mit 24 Bytes verwendet, während es sich bei JMSMessageID um eine Zeichenfolge handelt. 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 JMSMessageId-Feld festgelegt ist, wird überschrieben.

Wenn Sie die Möglichkeit benötigen, MQMD.MessageIDkönnen Sie dies mit einer der WebSphere MQ JMS-Erweiterungen tun, die in „Nachrichtendeskriptor in einer WebSphere MQ Classes for JMS-Anwendung lesen und schreiben” auf Seite 952beschrieben sind.

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. Weitere Informationen hierzu finden Sie im Abschnitt „Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker verwenden” auf Seite 981.

JMSCorrelationID zu MQMD.CorrelId, MQRFH2
JMSCorrelationID kann einen der folgenden Werte enthalten:

Eine providerspezifische Nachrichten-ID
Hierbei handelt es sich um eine Nachrichten-ID aus einer zuvor gesendeten oder empfangenen Nachricht. Daher sollte sie eine aus 48 Hexadezimalziffern Zeichenfolge in Kleinschreibung sein, die das Präfix *ID:* enthält. Das Präfix wird entfernt, die verbleibenden Zeichen werden in Binärdaten konvertiert und anschließend im Feld 'MQMD.CorrelId' festgelegt. 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 WebSphere 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. An-

dernfalls 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-providerspezifische Felder zuordnen
Die folgenden Hinweise beziehen sich auf die Zuordnung von JMS-Provider-spezifischen Feldern zu IBM WebSphere MQ-Nachrichten.

JMS_IBM_Report_<Name> zu MQMD-Bericht
Eine JMS-Anwendung kann die MQMD-Berichtsoptionen (Report) 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 WebSphere 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 WebSphere MQ-Warteschlangenziel gesetzt ist, wird MSGType auf den Wert MQMT_REQUEST gesetzt.
- Wenn JMSReplyTo nicht festgelegt oder auf einen anderen Wert als ein IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ-Anwendung anzuzeigen, dass diese Nachricht die letzte Nachricht in einer Gruppe ist. Beim Publish/Subscribe-Messaging wird diese Eigenschaft ignoriert.

Zuordnung von WebSphere MQ-Feldern zu JMS-Feldern (eingehende Nachrichten)
Diese Tabellen zeigen, wie JMS-Header- und -Eigenschaftsfelder bei der Ausführung von get()- oder receive()-Operationen MQMD- und MQRFH2-Feldern zugeordnet werden.

Tabelle 116 auf Seite 876 zeigt die Zuordnung der JMS-Headerfelder zu MQMD/MQRFH2-Feldern bei der Ausführung von get() oder receive().Tabelle 117 auf Seite 877 zeigt, wie JMS-Eigenschaftsfelder MQMD/MQRFH2 -Feldern beim Abrufen (get ()) oder Empfangen (receive ()) zugeordnet werden. Tabelle 118 auf Seite 877 zeigt, wie JMS-Provider-spezifische Eigenschaften zugeordnet werden.

Tabelle 116. Zuordnung der JMS-Headerfelder bei eingehenden Nachrichten

Name des JMS-Headerfelds	MQMD-Feld für Abruf	MQRFH2-Feld für Abruf
JMSDestination		jms.Dst oder mqps.Top „1" auf Seite 877

Tabelle 116. Zuordnung der JMS-Headerfelder bei eingehenden Nachrichten (Forts.)

Name des JMS-Headerfelds	MQMD-Feld für Abruf	MQRFH2-Feld für Abruf
JMSDeliveryMode	Persistence„2” auf Seite 877	jms.Dlv„2” auf Seite 877
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate„2” auf Seite 877 PutTime„2” auf Seite 877	jms.Tms„2” auf Seite 877
JMSCorrelationID	CorrelId„2” auf Seite 877	jms.Cid„2” auf Seite 877
JMSReplyTo	ReplyToQ„2” auf Seite 877 ReplyToQMgr„2” auf Seite 877	jms.Rto„2” auf Seite 877
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 Eigenschaftswert von JMS_IBM_Character_Set ist ein Zeichenfolgewert (String), der den Java-Zeichensatz enthält, der dem numerischen CodedCharacterSetId-Wert entspricht.

Tabelle 117. Eigenschaftszuordnung bei eingehenden Nachrichten

Name der JMS-Eigenschaft	MQMD-Feld für Abruf	MQRFH2-Feld für Abruf
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId„1” auf Seite 877	jms.Gid„1” auf Seite 877
JMSXGroupSeq	MsgSeqNumber„1” auf Seite 877	jms.Seq„1” auf Seite 877

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 118. Zuordnung der providerspezifischen JMS-Eigenschaften bei eingehenden Nachrichten

Name der JMS-Eigenschaft	MQMD-Feld für Abruf	MQRFH2-Feld für Abruf
JMS_IBM_Report_Exception	Bericht	

Tabelle 118. Zuordnung der providerspezifischen JMS-Eigenschaften bei eingehenden Nachrichten (Forts.)

Name der JMS-Eigenschaft	MQMD-Feld für Abruf	MQRFH2-Feld für Abruf
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 878	Encoding	
JMS_IBM_Character_Set „1" auf Seite 878	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 WebSphere MQ-Anwendung

In diesem Abschnitt wird beschrieben, was geschieht, wenn eine JMS-Anwendung Nachrichten mit einer konventionellen WebSphere MQ-Anwendung austauscht, die den MQRFH2-Header nicht verarbeiten kann.

. Abbildung 129 auf Seite 879 zeigt die Zuordnung.

Der Administrator gibt an, dass die JMS-Anwendung mit einer konventionellen WebSphere 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 WebSphere MQ-Anwendung als Ziel hat, ist mit der Zuordnung aus JMS zu einem MQMD identisch, der eine JMS-Anwendung als Ziel hat. Wenn WebSphere MQ Classes for JMS eine WebSphere MQ-Nachricht empfängt, 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 WebSphere MQ Classes for JMS eine Nachricht ohne MQRFH2-Header empfängt, 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.

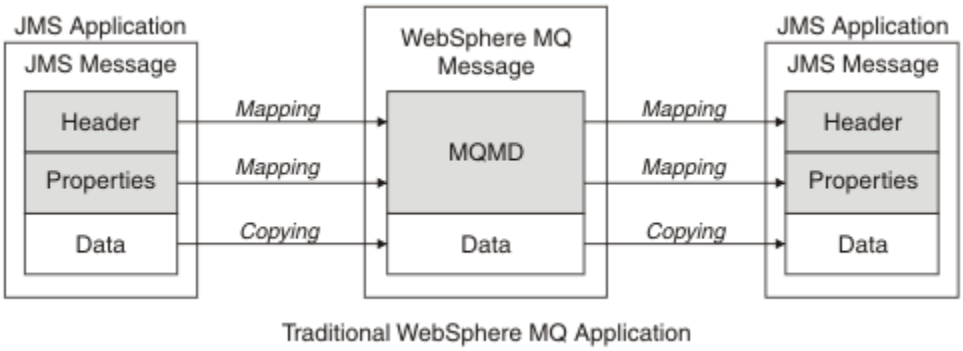


Abbildung 129. Transformation von JMS-Nachrichten in WebSphere MQ-Nachrichten ohne MQRFH2-Header

Der JMS-Nachrichtenhauptteil
Dieser Abschnitt enthält Informationen zur Codierung des Nachrichtenhauptteils selbst. Die Codierung hängt vom Typ der JMS-Nachricht ab.

ObjectMessage
Eine ObjectMessage ist ein Objekt, das von der Java Runtime auf normale Weise 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 WebSphere 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 WebSphere MQ-Header (bzw. im MQMD, falls keine Header vorhanden sind). Bei JMS-Nachrichten ist der letzte Header in der Regel der MQRFH2.

BytesMessage
Eine BytesMessage ist standardmäßig eine Bytefolge, die in der JMS-Spezifikation 1.0.2 und der zugehörigen Java-Dokumentation definiert 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 WebSphere 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. Dies ist im Java UTF8 -Format codiert und beginnt mit einem 2-Byte-Längenfeld. Die Zeichensatzeigenschaft des Zielobjekts hat keine Auswirkung auf die Codierung einer abgehenden BytesMessage. Der Zeichensatzwert in einer eingehenden WebSphere MQ-Nachricht hat keinen Einfluss auf die Interpretation dieser Nachricht als JMS-BytesMessage.

Nicht-Java-Anwendungen erkennen wahrscheinlich nicht die Java UTF8 -Codierung. 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 109 auf Seite 868 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 von WebSphere MQ Classes for JMS vor Version 5.3 wurde der Hauptteil einer Zuordnungsnachricht im folgenden Format codiert:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

WebSphere MQ Classes for JMS ab Version 5.3 kann jedes Format interpretieren, Versionen von WebSphere MQ Classes for JMS vor Version 5.3 können das aktuelle Format jedoch nicht interpretieren.

Wenn eine Anwendung Zuordnungsnachrichten an eine andere Anwendung senden muss, die eine Version von WebSphere MQ Classes for JMS vor Version 5.3 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 109 auf Seite 868 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. WebSphere 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 WebSphere 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 Nachrichten, die keine Textnachrichten sind, zwischen einer WebSphere 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. WebSphere MQ steuert die Codierung von Text und Zahlen in JMS-Nachrichten, außer bei `JMSObjectMessage` (siehe „JMSObjectMessage" auf Seite 888). Dabei werden drei Nachrichtenattribute verwendet. Die drei Attribute sind `CodedCharacterSetId`,`Encoding` und `Format`.

Diese drei Nachrichtenattribute werden normalerweise in den Feldern des JMS-Headers `MQRFH2` einer JMS-Nachricht gespeichert. Wenn der Nachrichtentyp nicht `JMS` sondern `MQ` lautet, werden die Attribute im Nachrichtendeskriptor `MQMD` gespeichert. Die Attribute werden verwendet, um die JMS-Nachrichtendaten zu konvertieren. JMS-Nachrichtendaten werden im Nachrichtendatenteil einer WebSphere MQ-Nachricht übertragen.

Eigenschaften von JMS-Nachrichten

Eigenschaften von JMS-Nachrichten, wie zum Beispiel `JMS_IBM_CHARACTER_SET`, werden im `MQRFH2`-Headerteil einer JMS-Nachricht ausgetauscht, es sei denn, die Nachricht wurde ohne einen `MQRFH2` gesendet. Nur `JMSTextMessage` und `JMSBytesMessage` können ohne einen `MQRFH2` gesendet werden. Wenn eine JMS-Eigenschaft als WebSphere MQ-Nachrichteneigenschaft im Nachrichtendeskriptor `MQMD` gespeichert ist, wird sie als Teil der `MQMD`-Konvertierung konvertiert. Wenn eine JMS-Eigenschaft im `MQRFH2` gespeichert ist, 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.

Konzepte für die 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 aus-

schließlich mit anderen JMS-Anwendungen austauscht, müssen Sie sich normalerweise keine Gedanken um die Datenkonvertierung machen. Die Datenkonvertierung wird automatisch von WebSphere MQ durchgeführt.

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 zwischen JMS-Anwendungen oder beim Austausch von Textnachrichten mit IBM WebSphere MQ-Programmen werden alle erforderlichen Konvertierungen automatisch von IBM WebSphere 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 882
2. „Anwendungsdatenkonvertierung” auf Seite 883
3. „Warteschlangenmanager-Datenkonvertierung” auf Seite 884
4. „Nachrichtenkanal-Datenkonvertierung” auf Seite 885

Wo sollte die Konvertierung stattfinden?
Im Abschnitt „Methode für die Nachrichtenkonvertierung wählen: receiver makes good” auf Seite 885 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

JMS-Client[4]Datenkonvertierung ist die Konvertierung von Java-Basiselementen und -Objekten in Bytes in einer JMS-Nachricht, wenn sie an ein Ziel gesendet wird, und umgekehrt, wenn sie empfangen wird. Die JMS-Client-Datenkonvertierung verwendet die Methoden der `JMSMessage`-Klassen. Die Methoden werden nach `JMSMessage`-Klassentyp in Tabelle 119 auf Seite 886 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 ebenfalls die Zielcodepage und die Codierung überschreiben, indem sie die Nachrichteneigenschaften festlegt, die das Schreiben oder Festlegen des Nachrichteninhalts steuern.

Wenn Sie beim Senden einer `JMSBytesMessage`-Nachricht an ein Ziel, dessen Codierung nicht als native Codierung (`Native`) definiert ist, die Zahlencodierung konvertieren möchten, müssen Sie die Nachrichteneigenschaft `JMS_IBM_ENCODING` vor dem Versenden der Nachricht festlegen. Falls Sie das Muster "receiver makes good" verwenden oder Nachrichten zwischen JMS-Anwendungen austauschen, muss die Anwendung `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 Anwendung finden soll.

[4] "JMS-Client" bezieht sich auf die WebSphere MQ Classes for JMS, die die JMS-Schnittstelle implementieren, die im Client-oder Bindungsmodus ausgeführt wird.

Zum Abruf der Daten in einer Nachricht ruft eine Anwendung die JMS-Methoden zum Lesen oder Abrufen von Nachrichten auf. Die Methoden beziehen sich auf die Codepage und die Codierung, die im vorherigen Nachrichtenheader definiert wurden, um die Java-Basiselemente und -Objekte ordnungsgemäß zu erstellen.

Die JMS-Client-Datenkonvertierung erfüllt die Anforderungen der meisten JMS-Anwendungen, die Nachrichten zwischen zwei JMS-Clients 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.

Weitere Informationen zur JMS-Client-Datenkonvertierung finden Sie im Abschnitt „Konvertierung und Codierung von JMS-Clientnachrichten” auf Seite 895.

Anwendungsdatenkonvertierung

Eine JMS-Clientanwendung kann eine explizite Zeichendatenkonvertierung mithilfe der Klasse `java.nio.charset.Charset` durchführen. Weitere Informationen finden Sie in den Beispielen unter Abbildung 132 auf Seite 887 und Abbildung 133 auf Seite 888 . 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. Normalerweise wird die Nachricht als `JMSBytesMessage`gesendet oder empfangen, weil der Nachrichtenteil eines `JMSBytesMessage` keine anderen Daten als die von der Anwendung geschriebenen Daten enthält.[5]. Sie können auch unter Verwendung von `JMSStreamMessage`, `JMSMapMessage` oder `JMSObjectMessage` Bytes senden und empfangen.

Es gibt keine Java-Methoden zum Codieren und Decodieren von Bytes, die numerische Daten enthalten, die 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 mit einer JMS-fremden Anwendung austauscht. Eine formatierte Nachricht enthält Text, numerische Daten und Bytedaten, die nach der Länge der Datenfelder organisiert sind. Wenn die JMS-fremde Anwendung nicht das Nachrichtenformat "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. Es gibt zwei Möglichkeiten, eine formatierte Nachricht in ein Java-Objekt zu konvertieren, das einfach zu verwenden ist.

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 903.

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

Warteschlangenmanager-Datenkonvertierung

In WebSphere MQ V7.0 kann durch den Warteschlangenmanager eine Codepagekonvertierung durchgeführt werden, wenn ein JMS-Clientprogramm eine Nachricht abruft. 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 131 auf Seite 887. 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, kann das JMS-Client-Programm auch die Zieleigenschaft festlegen; siehe Abbildung 130 auf Seite 884.

Vor V7.0 wurden 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 903. In V7.0 bis Fixpack 7.0.1.5 wird die Konvertierung immer vom Warteschlangenmanager vorgenommen, falls dies möglich ist. Ab 7.0.1.5 entspricht das Standardkonvertierungsverhalten wieder demjenigen der Version 6.0 und alle Konvertierungen werden vom JMS-Client durchgeführt. 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 130 auf Seite 884.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Oder

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Abbildung 130. 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 WebSphere MQ -Nachrichtentypen formatiert sind, wie z. B. einem CICS -Brückenheader (`MQCIH`). Wenn das Feld `Format` benutzerdefiniert ist, sucht der Warteschlangenmanager nach einem Datenkonvertierungsexit mit dem im Feld `Format` angegebenen Namen.

Die Warteschlangenmanager-Datenkonvertierung ist vor allem von Vorteil, wenn das Designmuster "receiver makes good" verwendet wird. Ein sendender JMS-Client muss keine Konvertierung durchführen. Ein empfangendes JMS-fremdes Programm verlässt sich auf den Konvertierungsexit, um sicherzustellen, dass die Nachricht in der erforderlichen Codepage und Codierung zugestellt wird. Bei einem sendenden JMS-Client und einem Empfänger, bei dem es sich nicht um einen JMS-Empfänger handelt, gilt das Beispiel für IBM WebSphere MQ-Versionen vor und nach V7.0. Bei IBM WebSphere MQ V7.0 kann 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 Satzformat erstellen, mit `com.ibm.mq.headers` als Java-Klasse darauf zugreifen und mit Ihrem eigenen Konvertierungsexit konvertieren. 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 903.

Nachrichtenkanal-Datenkonvertierung

Die Sender-, Server-, Clusterempfänger- und Clustersenderkanäle in WebSphere 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 WebSphere 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. Wenn der Empfänger ein JMS-Client ist, wird er trotzdem erneut konvertiert, um die Nachrichtendaten zu decodieren und Java-Basiselemente und -Objekte zu erstellen.

Der Unterschied zwischen JMS-Clientanwendungen und Anwendungen, die in einer Sprache wie C geschrieben sind, besteht darin, dass Java Datenkonvertierung durchführen muss. Eine Java-Anwendung muss Zahlen und Text aus ihrer 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 WebSphere MQ für die Codierung von Zahlen und Text in Nachrichten verwendet werden. Normalerweise übernehmen Sie den Zeichensatz `1208` und die Codierung `Native`.

WebSphere 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 unter `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage` oder `JMSObjectMessage` den 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 verhindern, dass die IBM WebSphere MQ-Klassen für 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 903. Dort sind Beispiele für die Konvertierung von Nachrichten enthalten, die mit JMS-fremden Anwendungen ausgetauscht werden.

Beispiele

Tabelle mit Nachrichtentypen und Konvertierungstypen

Tabelle 119. 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 119. 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

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 131. Codeausschnitt aus `amqsget0.c`

Text in einer `JMSBytesMessage` senden und empfangen

Der Code in Abbildung 132 auf Seite 887 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine `JMSTextMessage` geeigneter ist. Um eine Textzeichenfolge in Bytenachricht zu empfangen, die eine Mischung von Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen, die in Abbildung 133 auf Seite 888als *TEXT_LENGTH* bezeichnet wird. 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 132. 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 133. Zeichenfolge (`String`) von einer JMS`BytesMessage` empfangen

Zugehörige Konzepte

Konvertierung und Codierung von JMS-Clientnachrichten
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 V7.0 verwenden auch JMS-Clients, die Nachrichten empfangen, die Warteschlangenmanager-Datenkonvertierung. 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, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann, zu entwerfen und zu erstellen. 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 891 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 der Text Format auf "MQSTR ", WMQConstants.MQFMT_STRINGgesetzt. 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 WebSphere MQin CodedCharacterSetId codiert. Die Felder CodedCharacterSetId und Format werden entweder im Nachrichtendeskriptor MQMD oder in den JMS-Feldern in einem MQRFH2 festgelegt. Wenn die Nachricht mit dem Nachrichtenhauptteilstil WMQ_MESSAGE_BODY_MQ definiert oder der Hauptteilstil nicht angegeben ist, das Ziel jedoch WMQ_TARGET_DEST_MQ lautet, werden die Nachrichtendeskriptorfelder festgelegt. Andernfalls hat die Nachricht einen JMS-RFH2 und die Felder werden im festen Teil von MQRFH2 festgelegt.

Eine Anwendung kann die ID des codierten Zeichensatzes, die für ein Ziel definiert ist, überschreiben. Sie muss die Nachrichteneigenschaft JMS_IBM_CHARACTER_SET auf eine ID des codierten Zeichensatzes setzen; lesen Sie hierzu das Beispiel in „JMSTextmessage senden und empfangen" auf Seite 891.

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 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 891.

Eine JMSTextMessage kann in einem Nachrichtenhauptteil des MQ-Stils ohne den JMS-Header MQRFH2 gesendet werden. Die Werte der Zielattribute WMQ_MESSAGE_BODY und WMQ_TARGET_DEST bestimmen den Stil des Nachrichtenhauptteils, sofern sie nicht von der Anwendung überschrieben werden. Die Anwendung kann die im Ziel festgelegten Werte überschreiben, indem sie destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) oder destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) aufruft.

Wenn Sie eine JMSTextMessage mit einem Hauptteil im MQ-Stil senden, indem Sie sie an ein Ziel senden und dabei WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ setzen, können Sie sie nicht als JMSTextMessage von demselben Ziel empfangen. Alle von einem Ziel empfangenen Nachrichten, bei denen WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_MQ gesetzt ist, werden als JMSBytesMessage empfangen. Wenn Sie versuchen, die Nachricht als JMSTextMessage zu empfangen, führt dies zu einer Ausnahmebedingung: ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to javax.jms.TextMessage

Anmerkung: Der Text in einer JMSBytesMessage wird nicht durch den JMS-Client konvertiert. Der Client kann den Text in der Nachricht nur als Byte-Array empfangen. Wenn die Warteschlangenmanager-Konvertierung aktiviert ist, wird der Text vom Warteschlangenmanager konvertiert, der JMS-Client muss ihn aber dennoch als Byte-Array in einer JMSBytesMessage empfangen.

In der Regel ist es vorteilhafter, mit der Eigenschaft WMQ_TARGET_DEST zu steuern, ob eine JMSTextMessage mit einem Hauptteil im MQ- oder JMS-Stil gesendet wird. Anschließend können Sie die Nachricht 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 894. Details hierzu finden Sie unter „Konvertierung und Codierung von JMS-Clientnachrichten" auf Seite 895. Jedes primitive Element ist mit Tags versehen; siehe „Der JMS-Nachrichtenhauptteil" auf Seite 879.

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 138 auf Seite 892 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. Details finden Sie im Abschnitt Klasse JMSStreamMessage. Das Beispiel im Abschnitt „Daten in JMSStreamMessage und JMSMapMessage senden" auf Seite 892 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 894. Details hierzu finden Sie unter „JMS-Nachrichtentypen und Konvertierung" auf Seite 888.

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 WebSphere MQ Classes for JMS eine Konvertierung durchführt; siehe „Text in einer JMSBytesMessage senden und empfangen" auf Seite 892.

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. Code, der auf dem Beispiel „Nachrichten mithilfe von DataInputStream und DataOutputStream lesen und schreiben” auf Seite 893basiert, ist erforderlich, um das Paket com.ibm.mq.header mit JMS zu verwenden.

Beispiele

JMSObjectMessage senden und empfangen

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

Abbildung 134. 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 135. 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 136. 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 137. 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 138. Daten in `JMSStreamMessage` und `JMSMapMessage` senden

Text in einer `JMSBytesMessage` senden und empfangen

Der Code in Abbildung 139 auf Seite 892 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine `JMSTextMessage` geeigneter ist. Um eine Textzeichenfolge in Bytenachricht zu empfangen, die eine Mischung von Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen, die in Abbildung 140 auf Seite 893als *TEXT_LENGTH* bezeichnet wird. 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 139. 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 140. Zeichenfolge (String) von einer JMSBytesMessage empfangen

Nachrichten mithilfe von DataInputStream und DataOutputStream lesen und schreiben

Der Code in Abbildung 141 auf Seite 893 erstellt eine JMSBytesMessage mit einem DataOutput-Stream.

```
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 141. JMSBytesMessage mit DataOutputStream senden

Die Anweisung, die die Eigenschaft JMS_IBM_ENCODING festlegt, ist auskommentiert. Die Anweisung ist beim direkten Schreiben in eine JMSBytesMessage gültig, hat jedoch keine Auswirkung beim Schreiben in DataOutputStream. Die in den DataOutputStream geschriebenen Zahlen werden in der Codierung Native codiert. Die Festlegung von JMS_IBM_ENCODING hat keine Auswirkung.

Der Code in Abbildung 142 auf Seite 893 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 142. JMSBytesMessage mithilfe von DataInputStream empfangen

Die Codepage wird unter Verwendung der in den Eingabenachrichtendaten angegebenen Codepageeigenschaft JMS_IBM_CHARACTER_SET ausgegeben. Bei Eingabe ist JMS_IBM_CHARACTER_SET eine Java-Codepage und keine numerische ID des codierten Zeichensatzes.

Tabelle mit Nachrichtentypen und Konvertierungstypen

Tabelle 120. 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

Zugehörige Konzepte

Konzepte für die 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. Die Datenkonvertierung wird automatisch von WebSphere MQ durchgeführt.

Konvertierung und Codierung von JMS-Clientnachrichten
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 V7.0 verwenden auch JMS-Clients, die Nachrichten empfangen, die Warteschlangenmanager-Datenkonvertierung. 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, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann, zu entwerfen und zu erstellen. Der Austausch einer formatierten Nachricht mit einer JMS-fremden Anwendung kann mit oder ohne Aufruf eines Datenkonvertierungsexits erfolgen.

Konvertierung und Codierung von JMS-Clientnachrichten
Hier werden die Methoden aufgelistet, die Sie für die Nachrichtenkonvertierung und -codierung beim JMS-Client verwenden. Außerdem werden für jeden Konvertierungstyp Codebeispiele bereitgestellt.

Konvertierung und Codierung treten auf, wenn Java-Basiselemente oder -Objekte aus JMS-Nachrichten gelesen oder in JMS-Nachrichten 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.[6]in den Zeichensatz konvertiert, der für Text in Nachrichten verwendet wird. Numerische Daten werden in numerische Java-Basiselementtypen konvertiert und in die für die Nachricht definierte Codierung konvertiert. Ob die Konvertierung durchgeführt und welcher Konvertierungstyp dabei verwendet wird, hängt vom JMS-Nachrichtentyp sowie von der Lese- oder Schreiboperation ab.

In Tabelle 121 auf Seite 895 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 121. Nachrichtentypen und Konvertierungstypen

	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

[6] Bei manchen Unicode-Darstellungen sind mehr als 16 Bit erforderlich. Siehe eine Java SE-Referenz.

Tabelle 121. Nachrichtentypen und Konvertierungstypen (Forts.)				
	Konvertierungstyp			
Nachrichtentyp	**Text**	**Numerisch**	**Sonstiges**	**--**
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedS- hort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessa- ge	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.[7].

Bei den Codeausschnitten in „JMSTextmessage senden und empfangen" auf Seite 899 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.

[7] Beim Empfangen einer Nachricht ist JMS_IBM_CHARACTER_SET ein Java Charset -Codepagname.

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 147 auf Seite 899 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 WebSphere 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 146 auf Seite 899 mit dem falschen Codeausschnitt in Abbildung 143 auf Seite 897. Im falschen Ausschnitt wird die Textzeichenfolge zweimal konvertiert, und zwar einmal durch die Anwendung und nochmals durch WebSphere 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 143. 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 korrekt in numerische Java-Basiselemente umgewandelt. 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 von der 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 148 auf Seite 899 zeigt eine Anwendung, die eine Zahl im Zielformat codiert und diese in einer `JMSStreamMessage` sendet. Vergleichen Sie das Beispiel in Abbildung 148 auf Seite 899 mit dem Beispiel in Abbildung 149 auf Seite 900. Der Unterschied besteht darin, dass `JMS_IBM_ENCODING` in einer `JMSBytesMessage` festgelegt werden muss.

Anmerkung: Mit WebSphere 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 900 beschrieben.

WebSphere 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 WebSphere MQ -Klassen für Java verwenden JMS_IBM_CHARACTER_SET , um das Feld CodedCharacterSetId in den JMS-Eigenschaften im MQRFH2oder im Nachrichtendeskriptor MQMDfestzulegen:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);[8]
```

Wenn ein Byte-Array in eine JMSStringMessage oder JMSMapMessage geschrieben wird, führt WebSphere 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 144 auf Seite 899 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.

[8] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
           .getIntProperty(WMQConstants.WMQ_CCSID))));
```

Abbildung 144. Bytes, die eine Zeichenfolge in einer JMS`StreamMessage` darstellen, unter Verwendung des Zielzeichensatzes schreiben

Beispiele

JMS`Textmessage` 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 145. 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 146. 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 147. Textnachricht empfangen

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 148. Zahl unter Verwendung der Zielcodierung in einer JMS`StreamMessage` 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 149. Zahl unter Verwendung der Zielcodierung in einer JMSBytesMessage senden

Text in einer JMSBytesMessage senden und empfangen

Der Code in Abbildung 150 auf Seite 900 sendet eine Zeichenfolge in einer BytesMessage. Der Einfachheit halber wird in diesem Beispiel eine einzelne Zeichenfolge gesendet, für die eine JMSTextMessage geeigneter ist. Um eine Textzeichenfolge in Bytenachricht zu empfangen, die eine Mischung von Typen enthält, müssen Sie die Länge der Zeichenfolge in Bytes kennen, die in Abbildung 151 auf Seite 900als *TEXT_LENGTH* bezeichnet wird. 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 150. 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 151. Zeichenfolge (String) von einer JMSBytesMessage empfangen

Zugehörige Konzepte

Konzepte für die 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. Die Datenkonvertierung wird automatisch von WebSphere MQ durchgeführt.

Warteschlangenmanager-Datenkonvertierung
Die Warteschlangenmanager-Datenkonvertierung stand bereits in der Vergangenheit JMS-fremden Anwendungen zur Verfügung, die Nachrichten von JMS-Clients empfangen haben. Seit V7.0 verwenden auch JMS-Clients, die Nachrichten empfangen, die Warteschlangenmanager-Datenkonvertierung. 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, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von

JMSBytesMessage austauschen kann, zu entwerfen und zu erstellen. 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 V7.0 verwenden auch JMS-Clients, die Nachrichten empfangen, die Warteschlangenmanager-Datenkonvertierung. 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 konnte jedoch bis zu Version 7.0 nicht von einer JMS-Anwendung beim Empfang einer Nachricht verwendet werden.

In Versionen vor V7.0 können Sie die Warteschlangenmanager-Konvertierung in Verbindung mit 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.

Ab V7.0 kann 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. Sie entspricht dem Verhalten von WebSphere MQ V6.0, bei dem keine Warteschlangenmanager-Datenkonvertierung für JMS-Clients unterstützt wurde. 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 152. 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 decodieren Text in den empfangenen Daten aus UTF-8und erstellen primitive Java-Textelemente in ihrer internen 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 153. 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 154 auf Seite 902.

```
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 154. Codeausschnitt aus `amqsget0.c`

Anmerkung:

Die Warteschlangenmanager-Konvertierung erfolgt nur für die Nachrichtendaten, die ein bekanntes WebSphere 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

Konzepte für die 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. Die Datenkonvertierung wird automatisch von WebSphere MQ durchgeführt.

Konvertierung und Codierung von JMS-Clientnachrichten

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 443
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, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann, zu entwerfen und zu erstellen. 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, die Nachrichten mit einer JMS-fremden Anwendung unter Verwendung von `JMSBytesMessage` austauschen kann, zu entwerfen und zu erstellen. 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 WebSphere MQ für die Details der Nachrichtenformatierung zuständig. WebSphere 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 WebSphere 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 WebSphere MQ-Nachrichten” auf Seite 863.

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 907 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 WebSphere MQ-Headerklassen.

   Informationen zur Behandlung von vordefinierten WebSphere MQ-Headern finden Sie im Abschnitt Behandlung von WebSphere MQ-Nachrichtenheadern.

   Abbildung 155 auf Seite 905 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 907 und geben Sie dem Datenkonvertierungsexit den Namen `MYRECORD`.
3. Schreiben Sie Java-Klassen zum Einbinden des Datensatzlayouts und zum Senden und Empfangen von Datensätzen. 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 907.
   - 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.

   Siehe „Methode für die Nachrichtenkonvertierung wählen: receiver makes good" auf Seite 885.
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 907 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 156 auf Seite 906. 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 907.
   - Das WebSphere 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 155. 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 156. amqsget0.c ändern

Zugehörige Konzepte

Konzepte für die 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. Die Datenkonvertierung wird automatisch von WebSphere MQ durchgeführt.

Konvertierung und Codierung von JMS-Clientnachrichten

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 V7.0 verwenden auch JMS-Clients, die Nachrichten empfangen, die Warteschlangenmanager-Datenkonvertierung. 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.

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 der Task erstellen Sie einige Java-Klassen, um eine Beispielsatzstruktur in einem `JMSBytesMessage`auszutauschen. 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 WebSphere 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 Java-Programmierung und JMS vertraut sein. Es werden keine Anweisungen zum Einrichten 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. Das Beispiel erfordert IBM WebSphere MQ V7.0.
3. Das Beispiel wurde mithilfe der Java-Perspektive der Eclipse -Workbench erstellt. Es erfordert JRE 6.0 oder höher. Sie können die Java-Perspektive in IBM WebSphere MQ Explorer verwenden, um die

Java-Klassen zu entwickeln und auszuführen. Alternativ können Sie Ihre eigene Java-Entwicklungsumgebung verwenden.

4. Die Einrichtung der Testumgebung und das Debugging sind in IBM WebSphere 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 WebSphere MQ-Header modelliert, das eine Strukturkennung, eine Version und ein Längenfeld aufweist. Die Codierungs-, Zeichensatz- und Formatfelder, die häufig in IBM WebSphere 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 914 enthält eine umfassende Auflistung von RECORD und MyRecord. Darüber hinaus sind dort zusätzliche "Scaffolding"-Klassen für den Test von RECORD und MyRecord aufgelistet. Es gibt die folgenden zusätzlichen Klassen:

TryMyRecord
Das Hauptprogramm für den Test von RECORD und MyRecord.

EndPoint
Eine abstrakte Klasse, die die Verbindung, das Ziel und die Sitzung von JMS in eine einzelne Klasse einbindet. Ihre Schnittstelle erfüllt lediglich die Mindestanforderungen für das Testen der Klassen RECORD und MyRecord. Es handelt sich bei dieser Klasse nicht um ein etabliertes Designmuster für das Schreiben von JMS-Anwendungen.

Anmerkung: Die Endpoint-Klasse enthält nach der Erstellung eines Ziels folgende Codezeile:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

In V7.0 muss ab V7.0.1.5 die Warteschlangenmanager-Konvertierung aktiviert werden. Standardmäßig ist sie inaktiviert. In V7.0 bis V7.0.1.4 ist die Warteschlangenmanager-Konvertierung standardmäßig aktiviert und diese Codezeile führt zu einem Fehler.

MyProducer und MyConsumer
Klassen, die EndPoint und einen MessageConsumer und MessageProducer erstellen, die verbunden und zum Akzeptieren von Anforderungen bereit sind.

Gemeinsam bilden alle Klassen eine vollständige Anwendung, die Sie erstellen und mit der Sie experimentieren können, um sich mit der Datenkonvertierung in einer JMSBytesMessage vertraut zu machen.

Vorgehensweise

1. Erstellen Sie mit einem Standardkonstruktor eine abstrakte Klasse zur Einbindung der Standardfelder in einen IBM WebSphere MQ-Header. Später erweitern Sie die Klasse, um den Header genau an Ihre Anforderungen anzupassen.

```
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
```

Anmerkung:

a. Die Attribute `structID` bis `nextFormat` werden in der Reihenfolge aufgelistet, in der sie in einem IBM WebSphere 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 und IBM WebSphere MQ -Nachrichten verwenden IDs codierter Zeichensätze. Der Beispielcode verwendet Zeichensätze.

d. `int` wird von IBM WebSphere MQin `MQLONG` serialisiert. `MQLONG` enthält 4 Bytes.

2. Erstellen Sie die Getter und Setter für die privaten Attribute.

a) Erstellen oder generieren Sie die Getter:

```
    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }
```

b) Erstellen oder generieren Sie die Setter:

```
    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

3. Erstellen Sie einen Konstruktor, um eine RECORD-Instanz aus einer `JMSBytesMessage` zu erstellen.

```
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
```

```
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }
```

Anmerkung:

a. `messageCharset` und `messageEncoding` werden aus den Nachrichteneigenschaften erfasst, da sie die für das Ziel festgelegten Werte überschreiben. `format` wird nicht aktualisiert. In diesem Beispiel erfolgt keine Fehlerprüfung. Wenn der Konstruktor `Record(BytesMessage)` aufgerufen wird, wird vorausgesetzt, dass es sich bei `JMSBytesMessage` um eine Nachricht des Typs RECORD handelt. Die Linie "`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 IBM WebSphere MQ Classes for JMS beim Schreiben einer `JMSBytesMessage` nicht verwendet wird.

   "`messageCharset = myProducer.getCharset();`"-Aufrufe

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
    }
```

   Er ruft den Java-Zeichensatz aus einer ID des codierten Zeichensatzes ab.

"CCSID.getCodepage(ccsid)" befindet sich im Paket com.ibm.mq.headers. 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 WebSphere 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 WebSphere MQ-Hauptteilstil auf 'unspecified' (nicht angegeben) gesetzt wird, wenn eine Einstellung von JMS überschrieben werden muss.

Anmerkung:

IBM WebSphere MQ Classes for JMS schreibt 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 WebSphere 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. Im Gegensatz zu den JMS_IBM_*-Eigenschaften steuern die JMS_IBM_MQMD_*-Eigenschaften nicht, wie IBM WebSphere MQ Classes for JMS eine JMS-Nachricht erstellt. 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; }Â .
```

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 156 auf Seite 906 in „Formatierten Datensatz mit einer JMS-fremden Anwendung austauschen” auf Seite 903.
  - 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 WebSphere MQund führen Sie das Hauptprogramm TryMyRecordaus (siehe Abbildung 157 auf Seite 914).

1. Konfigurieren Sie IBM WebSphere MQ und JMS, um das Beispiel auszuführen. Mit den Anweisungen wird das Beispiel unter Windows ausgeführt.

   1. Erstellen Sie einen Warteschlangenmanager

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   2. Erstellen Sie eine Warteschlange

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   3. Erstellen Sie ein JNDI-Verzeichnis

   ```
   cd c:\
   md JNDI-Directory
   ```

   4. Wechseln Sie in das JMS-Verzeichnis 'bin'
   Das JMS-Verwaltungsprogramm muss dort ausgeführt werden. Der Pfad ist *MQ_INSTALLATI-ON_PATH*\java\bin.

   5. Erstellen Sie die folgenden JMS-Definitionen in der Datei 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
   ```

   6. 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 WebSphere MQ Explorer erstellen, ändern und durchsuchen.
3. Führen Sie TryMyRecord aus.

Für die Ausführung des Beispiels verwendete Klassen

Die in den Abbildungen Abbildung 157 auf Seite 914 bis Abbildung 162 auf Seite 918 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 157. 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 158. 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; }Â .


    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 159. 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 160. 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 161. 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 162. MyConsumer

Verbindungsfactorys und Ziele in einer Anwendung erstellen und konfigurieren, die die WebSphere MQ-Klassen für JMS verwendet

Eine Anwendung mit WebSphere MQ -Klassen für JMS kann Verbindungsfactorys und Ziele erstellen, indem sie sie als verwaltete Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) abruft, die JMS-Erweiterungen von IBM oder die JMS-Erweiterungen von WebSphere MQ verwendet. Eine Anwendung kann die IBM JMS-Erweiterungen oder WebSphere 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 mit dem WebSphere MQ-JMS-Verwaltungstool oder mit WebSphere MQ Explorer Verbindungsfactorys und Ziele als verwaltete Objekte in einem JNDI-Namensbereich 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 WebSphere 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.

WebSphere MQ-JMS-Erweiterungen verwenden
Eine Anwendung kann auch die WebSphere MQ-JMS-Erweiterungen verwenden, um dynamisch zur Laufzeit Verbindungsfactorys und Ziele zu 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.

JNDI zum Abrufen verwalteter Objekte in einer JMS-Anwendung verwenden

Um verwaltete Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) abzurufen, muss eine JMS-Anwendung einen Ausgangskontext erstellen und anschließend die Methode lookup () verwenden, um die Objekte abzurufen.

Bevor eine Anwendung verwaltete Objekte aus einem JNDI-Namensbereich abrufen kann, muss ein Administrator diese verwalteten Objekte zuerst erstellen. Der Administrator kann das WebSphere MQ JMS-Verwaltungstool oder WebSphere MQ Explorer zum Erstellen und Verwalten von verwalteten Objekten in einem JNDI-Namensbereich verwenden. Informationen zur Verwendung des WebSphere MQ JMS-Verwaltungstools erhalten Sie unter „Einsatz des WebSphere MQ JMS-Verwaltungstools” auf Seite 992. Informationen zur Verwendung von WebSphere MQ Explorer finden Sie in der bei WebSphere MQ Explorer bereitgestellten Hilfe. Allerdings stellt ein Anwendungsserver in der Regel ein eigenes Repository für verwaltete Objekte und eigene Tools zum Erstellen und Pflegen dieser Objekte bereit.

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

IBM JMS-Erweiterungen verwenden

WebSphere MQ Classes for JMS enthält 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 von WebSphere MQ Classes for JMS-Objekten 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 den folgenden Paketen:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Die Pakete befinden sich in der Datei `com.ibm.mqjms.jar` im Verzeichnis `<MQ_Install_Dir>/java/lib`.

Diese Erweiterungen stellen die folgende Funktion bereit:

- Ein Factory-basierter Mechanismus zum dynamischen Erstellen von Verbindungsfactorys und Zielen zur Laufzeit, anstatt sie als verwaltete Objekte aus einem JNDI-Namensbereich (Java Naming and Directory Interface) abzurufen.
- Eine Gruppe von Methoden für die Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten
- Eine Gruppe mit Ausnahmebedingungsklassen, die Methoden für den Abruf detaillierter Informationen zu einem Problem enthalten
- Eine Gruppe von Methoden für die Steuerung der Traceverarbeitung
- Eine Gruppe von Methoden für den Abruf der Versionsinformationen zu WebSphere 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 WebSphere MQ-JMS-Erweiterungen bereit. Während die WebSphere MQ-JMS-Erweiterungen jedoch nur für den WebSphere-Messaging-Provider vorgesehen sind, sind die IBM JMS-Erweiterungen nicht WebSphere MQ-spezifisch, sondern können zusammen mit jedem beliebigen Messaging-Provider innerhalb der Schichtarchitektur verwendet werden, die im Abschnitt „Schichtarchitektur” auf Seite 849 beschrieben ist.

Die Schnittstelle com.ibm.msg.client.wmq.WMQConstants enthält die Definitionen von Konstanten, die eine Anwendung beim Festlegen der Eigenschaften von WebSphere MQ Classes for JMS-Objekten unter Verwendung der IBM JMS-Erweiterungen nutzen kann. Die Schnittstelle enthält Konstanten für den WebSphere 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 WebSphere 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 „Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten” auf Seite 922.

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 WebSphere 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 WebSphere MQ-Warteschlange und optional den Namen des Warteschlangenmanagers, der Eigner der Warteschlange

ist, und eine oder mehrere Eigenschaften des JmsQueue-Objekts angibt. 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 Objekt JmsQueue stellt eine WebSphere MQ -Warteschlange mit dem Namen Q2 dar, deren Eigner der Warteschlangenmanager QM2ist. 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 935. Eine alternative Methode für das Festlegen der Eigenschaften eines JmsQueue-Objekts ist im Abschnitt „Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten" auf Seite 922 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 935. Eine alternative Methode für das Festlegen der Eigenschaften eines JmsTopic-Objekts ist im Abschnitt „Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten" auf Seite 922 beschrieben.

Nachdem eine Anwendung eine Verbindungsfactory oder ein Ziel erstellt hat, kann dieses Objekt nur mit dem ausgewählten Messaging-Provider verwendet werden.

Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten

Zur Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten mithilfe der IBM JMS-Erweiterungen verwendet eine Anwendung die Methoden der Schnittstelle com.ibm.msg.client.JmsPropertyContext.

Für jeden Java-Datentyp enthält die Kontextschnittstelle JmsPropertyeine Methode zum Festlegen des Werts einer Eigenschaft mit diesem Datentyp und eine Methode zum Abrufen 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 JmsPropertyContext-Schnittstelle. Daher kann eine Anwendung diese Methoden zum Festlegen der Eigenschaften von MQConnectionFactory-, MQQueue- und MQTopic-Objekten verwenden.

Wenn eine Anwendung ein WebSphere MQ Classes for JMS-Objekt 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, löst WebSphere 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. WebSphere MQ Classes for JMS löst die Ausnahmebe-

dingung 'NumberFormatException' für primitive Datentypen aus und gibt für referenzierte Datentypen 'null' zurück.

Neben den vordefinierten Eigenschaften eines WebSphere MQ Classes for JMS-Objekts kann eine Anwendung eigene Eigenschaften festlegen. Diese anwendungsdefinierten Eigenschaften werden von WebSphere MQ Classes for JMS ignoriert.

Sie finden weitere Informationen zu den Eigenschaften von WebSphere MQ Classes for JMS-Objekten im Abschnitt Eigenschaften von IBM WebSphere MQ classes for JMS-Objekten.

Der folgende Code zeigt an einem Beispiel, wie mit den IBM JMS-Erweiterungen Eigenschaften festgelegt werden. 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 WebSphere MQ Classes for JMS-Objekts 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 Konvertierungen, die durch WebSphere MQ Classes for JMS unterstützt werden, sind in Tabelle 122 auf Seite 924 aufgeführt.

Tabelle 122. 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, löst WebSphere MQ Classes for JMS die Ausnahmebedingung 'NumberFormatException' aus.
- Wenn eine Anwendung eine nicht unterstützte Konvertierung versucht, löst WebSphere 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, löst WebSphere 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, löst WebSphere 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 „Festlegung der Eigenschaften von WebSphere MQ Classes for JMS-Objekten” auf Seite 922 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 WebSphere MQ Classes for JMS-Objekten 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 durch das WebSphere 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 speichert WebSphere 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, muss WebSphere MQ Classes for JMS die Namen und Werte aus dem angegebenen Eingabeformat in das kanonische Format konvertieren. Ähnlich gilt, dass WebSphere 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 muss, 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 WebSphere MQ Classes for JMS-Protokoll 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 WebSphere MQ Classes for JMS-Objekts zuzugreifen.

Der folgende Code gibt beispielsweise die Namen und Werte von allen Eigenschaften einer Verbindungsfactory aus. Der Code verwendet nur die Methoden der Map-Schnittstelle, um die Namen und Werte der Eigenschaften abzurufen.

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Prüfungen oder Konvertierungen von Eigenschaften können nicht mit den Methoden der Map-Schnittstelle umgangen werden.

WebSphere MQ-JMS-Erweiterungen verwenden

WebSphere MQ Classes for JMS enthält eine Gruppe von Erweiterungen für die JMS-API. Diese werden als 'WebSphere 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.

WebSphere MQ Classes for JMS enthält eine Gruppe von Klassen in den Paketen 'com.ibm.jms' und 'com.ibm.mq.jms'. Diese Klassen implementieren die JMS-Schnittstellen und enthalten die WebSphere MQ-JMS-Erweiterungen. 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.*;
```

Eine Anwendung kann die WebSphere MQ-JMS-Erweiterungen verwenden, um die folgenden Funktionen auszuführen:

- Erstellen Sie Verbindungsfactorys und Ziele dynamisch zur Laufzeit, anstatt sie als verwaltete Objekte aus einem JNDI-Namensbereich (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 928.

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 WebSphere MQ Classes for JMS-Objekten .

Weitere Informationen zu den Eigenschaften einer Verbindungsfactory und den Methoden, die für deren Festlegung verwendet werden, finden Sie im Abschnitt Eigenschaften von IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ-Warteschlange und optional den Namen des Warteschlangenmanagers, der Eigner der Warteschlange ist, und eine oder mehrere Eigenschaften des MQQueue-Objekts angibt. 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 WebSphere MQ-Warteschlange mit dem Namen Q3 dar. Der Eigner dieser Warteschlange ist der Warteschlangenmanager QM3 und 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 935. Eine alternative Methode für das Festlegen der Eigenschaften eines MQQueue-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen” auf Seite 930 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 935. Eine alternative Methode für das Festlegen der Eigenschaften eines MQTopic-Objekts ist im Abschnitt „Eigenschaften von Zielen festlegen” auf Seite 930 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 WebSphere 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, erstellt IBM WebSphere MQ classes for JMS eine Verbindungskennung (Hconn) und startet 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 'create-Connection(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, erstellt IBM WebSphere MQ classes for JMS eine Verbindungskennung (Hconn) und startet 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 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

Anstatt Ziele als verwaltete Objekte aus einem JNDI-Namensbereich abzurufen, kann eine JMS-Anwendung eine Sitzung verwenden, um Ziele dynamisch zur Laufzeit zu erstellen. Eine Anwendung kann einen Uniform Resource Identifier (URI) verwenden, um eine WebSphere MQ-Warteschlange oder ein Thema und optional eine oder mehrere Eigenschaften eines Queue- oder Topic-Objekts anzugeben.

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 WebSphere 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 WebSphere MQ-Warteschlange und optional den Namen des Warteschlangenmanagers, der Eigner der Warteschlange ist, und eine oder mehrere Eigenschaften des Queue-Objekts angibt. Die folgende Anweisung enthält ein Beispiel für einen Warteschlangen-URI:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

Das mit dieser Anweisung erstellte Warteschlangenobjekt stellt eine WebSphere MQ -Warteschlange mit dem Namen Q2 dar, deren Eigner ein Warteschlangenmanager mit dem Namen QM2ist. 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 Warteschlangenmana-

ger handelt, muss WebSphere 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 935.

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, erstellt WebSphere 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 935.

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 Topic erstellt, erstellt WebSphere MQ Classes for JMS ein Topic mit einem Namen, der mit den Zeichen TEMP/*tempTopicPräfix*beginnt, wobei *tempTopicPräfix* der Wert der Eigenschaft TEMPTOPICPREFIX der Verbindungsfactory ist.

Uniform Resource Identifiers (URIs)

Ein Warteschlangen-URI ist eine Zeichenfolge, die den Namen einer WebSphere MQ-Warteschlange und optional den Namen des Warteschlangenmanagers, der Eigner der Warteschlange ist, und eine oder mehrere Eigenschaften des Queue-Objekts, das durch die Anwendung erstellt wird, angibt. Ein Themen-URI ist eine Zeichenfolge, die den Namen eines Themas und optional eine oder mehrere Eigenschaften des von der Anwendung erstellten Topic-Objekts angibt.

Ein Warteschlangen-URI hat folgendes Format:

```
queue://[qMgrName]/qName[?propertyName1=propertyValue1
                         &propertyName2=propertyValue2
                         &...]
```

Ein Themen-URI hat folgendes Format:

```
topic://topicName[?propertyName1=propertyValue1
                  &propertyName2=propertyValue2
                  &...]
```

Die Variablen in diesen Formaten haben folgende Bedeutungen:

qMgrName
Der Name des Warteschlangenmanagers, der Eigner der Warteschlange ist, die durch den URI angegeben ist.

Bei dem Warteschlangenmanager kann es sich um den lokalen Warteschlangenmanager oder um einen fernen Warteschlangenmanager handeln. Wenn es sich um einen fernen Warteschlangenmanager handelt, muss WebSphere MQ so konfiguriert sein, dass WebSphere MQ die Nachricht vom lokalen Warteschlangenmanager an den fernen Warteschlangenmanager weiterleiten kann, wenn eine Anwendung eine Nachricht an diese Warteschlange sendet.

Erfolgt keine Angabe des Namens, wird der lokale Warteschlangenmanager verwendet.

qName
Der Name der WebSphere MQ-Warteschlange.

Bei der Warteschlange kann es sich um eine lokale Warteschlange, um eine Aliaswarteschlange oder um eine Definition einer fernen Warteschlange handeln.

Sie finden die Regeln für die Erstellung von Warteschlangennamen im Abschnitt Regeln für die Benennung von IBM WebSphere MQ-Objekten.

topicName
Der Name des Themas.

Sie finden die Regeln für die Erstellung von Themennamen im Abschnitt Regeln für die Benennung von IBM WebSphere MQ-Objekten. Vermeiden Sie die Verwendung der Platzhalterzeichen +, #, * und? in Themennamen. Themennamen, die diese Zeichen enthalten, können zu unerwarteten Ergebnissen führen, wenn Sie diese subskribieren. Siehe Themenzeichenfolgen verwenden.

***propertyName1*, *propertyName2*, ...**
Die Namen der Eigenschaften des Queue- oder Topic-Objekts, das durch die Anwendung erstellt wird. In Tabelle 123 auf Seite 936 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 123 auf Seite 936 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 123 auf Seite 936 sind die gültigen Eigenschaftsnamen und gültigen Werte aufgelistet, die in Warteschlangen- und Themen-URIs verwendet werden können. Das WebSphere MQ-JMS-Verwaltungstool verwendet zwar symbolische Konstanten für die Werte von Eigenschaften, URIs können jedoch keine symbolischen Konstanten enthalten.

Tabelle 123. 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 WebSphere MQ Classes for JMS die Nachricht an ein Ziel weiterleitet	• Eine beliebige ID des codierten Zeichensatzes, die von WebSphere MQ unterstützt wird.
encoding	Gibt an, wie die numerischen Daten im Hauptteil einer Nachricht dargestellt werden, wenn WebSphere MQ Classes for JMS die Nachricht an ein Ziel weiterleitet	• Ein beliebiger gültiger Wert für das Feld *Encoding* in einem WebSphere 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.

Tabelle 123. Eigenschaftsnamen und gültige Werte für die Verwendung in Warteschlangen- und Themen-URIs (Forts.)

Eigenschaftsname	Beschreibung	Gültige Werte
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 WebSphere MQ-JMS-Verwaltungstool verwendet wird. Sie finden eine Beschreibung der MULTICAST-Eigenschaft und ihrer gültigen Werte im Abschnitt Eigenschaften von IBM WebSphere 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 WebSphere MQ angegeben. • 1 - Nicht persistent. • 2 - Persistent. • 3 - Äquivalent zu dem Wert HIGH für die Eigenschaft PERSISTENCE, wie im WebSphere MQ-JMS-Verwaltungstool verwendet. Sie finden eine Erläuterung dieses Werts im Abschnitt „Persistente JMS-Nachrichten” auf Seite 961.
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 WebSphere 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 WebSphere 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 WebSphere MQ-Warteschlange mit dem Namen Q2 an, deren Eigner ein Warteschlangenmanager mit dem Namen QM2 ist. Alle Nachrichten, die an dieses Ziel gesendet werden, haben die Priorität 6. Die verbleibenden Eigenschaften des Queue-Objekts, das mit diesem URI erstellt wird, haben die jeweiligen Standardwerte.

```
queue://QM2/Q2?priority=6
```

Der folgende URI gibt ein Thema mit dem Namen Sport/Athletics/Results an. Alle Nachrichten, die an dieses Ziel gesendet werden, sind persistent und haben die Priorität 0. Die verbleibenden Eigenschaften des Topic-Objekts, das mit diesem URI erstellt wird, haben die jeweiligen Standardwerte.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Nachrichten in einer JMS-Anwendung senden

Bevor eine JMS-Anwendung Nachrichten an ein Ziel senden kann, muss sie ein MessageProducer-Objekt für das Ziel erstellen. Um eine Nachricht an das Ziel zu senden, erstellt die Anwendung ein Message-Objekt und ruft dann die send()-Methode des MessageProducer-Objekts auf.

Eine Anwendung verwendet ein MessageProducer-Objekt zum Senden von Nachrichten. Normalerweise erstellt eine Anwendung ein MessageProducer-Objekt für ein bestimmtes Ziel, das eine Warteschlange oder ein Thema sein kann, damit alle Nachrichten, die mit dem Nachrichtenproduzenten gesendet werden, an dasselbe Ziel gesendet werden. Daher muss zuerst ein Queue- oder Topic-Objekt erstellt werden, damit eine Anwendung ein MessageProducer-Objekt erstellen kann. Sie finden Informationen zur Vorgehensweise bei der Erstellung eines Queue- oder Topic-Objekts in den folgenden Abschnitten:

- „JNDI zum Abrufen verwalteter Objekte in einer JMS-Anwendung verwenden” auf Seite 919
- „IBM JMS-Erweiterungen verwenden” auf Seite 920
- „WebSphere MQ-JMS-Erweiterungen verwenden” auf Seite 927
- „Ziele in einer JMS-Anwendung erstellen” auf Seite 933

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 859.

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 WebSphere MQ Classes for JMS die Nachricht weiterleitet und die Steuerung an die Anwendung zurückgibt, ohne zu ermitteln, ob der Warteschlangenmanager die Nachricht fehlerfrei erhalten hat, wenn eine Anwendung Nachrichten an dieses Ziel sendet. Dies wird gelegentlich als *asynchrone Put-Operation* bezeichnet. Weitere Informationen finden Sie unter „Nachrichten in IBM WebSphere MQ Classes for JMS asynchron einreihen” auf Seite 978.

Nachrichten in einer JMS-Anwendung empfangen

Eine Anwendung verwendet einen Nachrichtenkonsumenten für das Empfangen von Nachrichten. Ein permanenter Topic-Subskribent ist ein Nachrichtenkonsument, der alle Nachrichten empfängt, die an ein Ziel gesendet werden. Dies betrifft auch Nachrichten, die gesendet werden, solange der Konsument inaktiv ist. Eine Anwendung kann mithilfe eines Nachrichtenselektors auswählen, welche Nachrichten sie empfangen möchte, und sie kann unter Verwendung eines Nachrichtenlisteners Nachrichten asynchron empfangen.

Eine Anwendung verwendet ein MessageConsumer-Objekt zum Empfangen von Nachrichten. Eine Anwendung erstellt ein MessageConsumer-Objekt für ein bestimmtes Ziel, das eine Warteschlange oder ein Thema sein kann, damit alle Nachrichten, die mit dem Nachrichtenkonsumenten empfangen werden, aus demselben Ziel empfangen werden. Daher muss zuerst ein Queue- oder Topic-Objekt erstellt werden, damit eine Anwendung ein MessageConsumer-Objekt erstellen kann. Sie finden Informationen zur Vorgehensweise bei der Erstellung eines Queue- oder Topic-Objekts in den folgenden Abschnitten:

- „JNDI zum Abrufen verwalteter Objekte in einer JMS-Anwendung verwenden” auf Seite 919
- „IBM JMS-Erweiterungen verwenden” auf Seite 920
- „WebSphere MQ-JMS-Erweiterungen verwenden” auf Seite 927
- „Ziele in einer JMS-Anwendung erstellen” auf Seite 933

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 WebSphere MQ Classes for JMS empfängt ein nicht permanenter Topic-Subskribent auch ständige Veröffentlichungen. Dies gilt aber nicht, wenn eine Echtzeitverbindung mit einem Broker verwendet wird.

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 WebSphere 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 860.

Das folgende Beispiel zeigt, wie eine Anwendung Nachrichten auf Basis einer benutzerdefinierten Eigenschaft mit dem Namen 'myProp' auswählen kann:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

Die JMS-Spezifikation erlaubt einer Anwendung nicht, den Nachrichtenselektor eines Nachrichtenkonsumenten zu ändern. Nachdem eine Anwendung einen Nachrichtenkonsumenten mit einem Nachrichtenselektor erstellt hat, bleibt der Nachrichtenselektor für den Lebenszyklus dieses Konsumenten erhalten. Falls eine Anwendung mehrere Nachrichtenselektoren benötigt, muss die Anwendung für jeden einzelnen Nachrichtenselektor einen Nachrichtenkonsumenten erstellen.

Beachten Sie, dass die Eigenschaft MSGSELECTION der Verbindungsfactory keine Auswirkung hat, wenn eine Anwendung mit einem Warteschlangenmanager der Version 7 verbunden ist. Zur Leistungsoptimierung erfolgt die gesamte Nachrichtenauswahl durch den Warteschlangenmanager.

Lokale Veröffentlichungen unterdrücken

Eine Anwendung kann einen Nachrichtenkonsumenten erstellen, der Veröffentlichungen ignoriert, die in der eigenen Verbindung des Konsumenten veröffentlicht werden. Hierfür setzt die Anwendung den dritten Parameter in einem createConsumer()-Aufruf auf `true`, wie im folgenden Beispiel dargestellt:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

Bei einem createDurableSubscriber()-Aufruf setzt die Anwendung hierfür wie im folgenden Beispiel den vierten Parameter auf`true`:

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Asynchrone Nachrichtenübermittlung

Eine Anwendung kann Nachrichten asynchron empfangen, indem sie einen Nachrichtenlistener bei einem Nachrichtenkonsumenten registriert. Der Nachrichtenlistener verfügt über eine Methode mit dem Namen 'onMessage', die asynchron aufgerufen wird, sobald eine geeignete Nachricht verfügbar ist. Ihr Zweck besteht in der Verarbeitung der Nachricht. Der folgende Code veranschaulicht das Verfahren:

```
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)
- 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 asynchronously:'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 „Handhabung nicht verarbeitbarer Nachrichten in IBM WebSphere MQ classes for JMS" auf Seite 945.

V 7.5.0.8 Abruf von Subskriptionsbenutzerdaten

Wenn die Nachrichten, die eine IBM WebSphere MQ classes for JMS-Anwendung aus einer Warteschlange liest, von einer administrativ definierten, permanenten Subskription eingereiht werden, muss die Anwendung auf die Benutzerdateninformationen, die der Subskription zugeordnet sind, zugreifen. Diese Informationen werden als Eigenschaft zur Nachricht hinzugefügt.

Beim Auslesen einer Nachricht mit einem MQPS-Ordner im RFH2-Header aus einer Warteschlange wird ab Version 7.5.0, Fix Pack 8 der dem Schlüssel 'Sud' (sofern vorhanden) zugeordnete Wert als Eigenschaft des Typs 'String' dem JMS-Nachrichtenobjekt hinzugefügt, das an die Anwendung, die die IBM WebSphere MQ classes for JMS verwendet, 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.

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 Anwendung, die die IBM WebSphere MQ classes for JMS verwendet, kann Folgendes aufrufen:

```
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

„Der MQRFH2-Header und JMS" auf Seite 864

Zugehörige Tasks

Verwaltungssubskription definieren

Zugehörige Verweise

SUB DEFINI

Schnittstelle JmsConstants

WebSphere MQ Classes for JMS schließen

Bei einer WebSphere MQ Classes for JMS-Anwendung 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 WebSphere MQ Classes for JMS- und WebSphere 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.

Handhabung nicht verarbeitbarer Nachrichten in IBM WebSphere MQ classes for JMS

Eine nicht verarbeitbare Nachricht ist eine Nachricht, die nicht von einer empfangenden MDB-Anwendung verarbeitet werden kann. Wenn eine nicht verarbeitbare Nachricht festgestellt wird, kann sie von den JMS-Objekten 'MessageConsumer' und 'ConnectionConsumer' in Übereinstimmung mit den beiden Warteschlangeneigenschaften BOQNAME und BOTHRESH neu eingereiht werden.

Manchmal kommt eine fehlerhaft formatierte Nachricht in einer Warteschlange an. In diesem Kontext bedeutet 'fehlerhaft formatiert', dass die empfangende Anwendung die Nachricht nicht ordnungsgemäß verarbeiten kann. Eine solche Nachricht kann dazu führen, dass die empfangende Anwendung fehlschlägt und diese fehlerhaft formatierte Nachricht zurücksetzt. Die Nachricht kann dann mehrfach an die Eingabewarteschlange übermittelt und mehrfach von der Anwendung zurückgesetzt werden. Diese Nachrichten werden als *nicht verarbeitbare Nachrichten* bezeichnet. Das JMS-Objekt 'MessageConsumer' erkennt nicht verarbeitbare Nachrichten und leitet diese an ein alternatives Ziel um.

Der IBM WebSphere MQ-Warteschlangenmanager zeichnet die Häufigkeit auf, mit der die einzelnen Nachrichten zurückgesetzt wurden. Sobald diese Anzahl einen konfigurierbaren Schwellenwert erreicht, reiht der Nachrichtenkonsument die Nachricht in eine namentlich genannte Rücksetzwarteschlange ein. Wenn diese Neueinreihung aus irgendeinem Grund fehlschlägt, wird die Nachricht aus der Eingabewarteschlange entfernt und entweder neu in die Warteschlange für nicht zustellbare Nachrichten eingereiht oder gelöscht. Weitere Informationen hierzu finden Sie unter „Nachrichten aus der Warteschlange in ASF entfernen” auf Seite 987.

Nicht verarbeitbare Nachrichten werden von MessageConsumers und ConnectionConsumers unterschiedlich neu eingereiht. ConnectionConsumers können nicht verarbeitbare Nachrichten erneut in die Warteschlange stellen, ohne dass sich dies auf die Nachrichtenübermittlung auswirkt. Der Neueinreihungsprozess findet außerhalb einer Arbeitseinheit statt, die der eigentlichen Nachrichtenübermittlung und dem entsprechenden Anwendungscode zugeordnet ist. Dies ist aufgrund der Multithread-Beschaffenheit der ConnectionConsumer-Operation möglich.

MessageConsumers werden jedoch als Einzelthreads unter der Sitzungsebene (Session) ausgeführt und jede Neueinreihung von nicht verarbeitbaren Nachrichten findet innerhalb der aktuellen Arbeitseinheit statt. Dies hat keine Auswirkung auf den Betrieb der Anwendung. Wenn nicht verarbeitbare Nachrichten jedoch unter einer transaktionsbasierten Sitzung oder einer Client_acknowledge-Sitzung neu eingereiht werden, wird die Neueinreihungsaktion selbst erst dann festgeschrieben, wenn die aktuelle Arbeitseinheit

durch den Anwendungscode oder gegebenenfalls durch den Anwendungscontainercode festgeschrieben wird.

JMS-ConnectionConsumer-Objekte handhaben nicht verarbeitbare Nachrichten auf dieselbe Weise und mit denselben Warteschlangeneigenschaften. Wenn mehrere Verbindungskonsumenten dieselbe Warteschlange überwachen, kann es vorkommen, dass die nicht verarbeitbare Nachricht häufiger an eine Anwendung übermittelt wird, als durch den Schwellenwert festgelegt, bevor die Neueinreihung durchgeführt wird. Dieses Verhalten ist auf die Art und Weise zurückzuführen, in der einzelne Verbindungskonsumenten Warteschlangen überwachen und nicht verarbeitbare Nachrichten erneut in die Warteschlange stellen.

Der Schwellenwert und der Name der Rücksetzwarteschlange sind Attribute einer IBM WebSphere MQ-Warteschlange. Diese Attribute haben die Namen 'BackoutThreshold' und 'BackoutRequeueQName'. Sie werden auf folgende Warteschlangen angewendet:

- Beim Punkt-zu-Punkt-Messaging handelt es sich um die zugrunde liegende lokale Warteschlange. Dies ist wichtig, wenn Nachrichtenkonsumenten und Verbindungskonsumenten Aliasnamen einer Warteschlange verwenden.
- Beim Publish/Subscribe-Messaging im normalen Modus des IBM WebSphere MQ-Messaging-Providers wird die verwaltete Warteschlange des Themas auf Basis der Modellwarteschlange erstellt.
- Beim Publish/Subscribe-Messaging im Migrationsmodus des IBM WebSphere MQ-Messaging-Providers gelten sie für die CCSUB-Warteschlange, die im TopicConnectionFactory-Objekt definiert ist, oder für die CCDSUB-Warteschlange, die im Topic-Objekt definiert ist.

IBM WebSphere MQ classes for JMS fragen die Attribute BackoutThreshold und BackoutRequeueQName der Warteschlange ab. Sie müssen daher dem Benutzer, der die Anwendung ausführt, Abfragezugriff auf die Warteschlange erteilen.

V 7.5.0.9 Wenn es sich bei der Zielwarteschlange um eine Clusterwarteschlange handelt, hängen die erforderlichen Berechtigungen von der verwendeten IBM WebSphere MQ classes for JMS -Version ab:

- Wenn Sie IBM WebSphere MQ classes for JMS für Version 7.5.0, Fix Pack 9 und einen vorläufigen Fix für APAR IT26482verwenden, ist der Abfragezugriff erforderlich.
- Erteilen Sie für alle anderen Versionen die Berechtigung 'inquire', 'browse' und 'get access '.

Geben Sie folgenden MQSC-Befehl aus, um die Attribute BackoutThreshold und BackoutRequeueQName festzulegen:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

Wenn das BackoutThreshold-Attribut auf einen anderen Wert als null gesetzt ist, setzen Sie das BackoutRequeueQName-Attribut auf einen gültigen Warteschlangennamen, um ein nicht erwartetes Verhalten zu vermeiden.

Wenn Ihr System beim Publish/Subscribe-Messaging für jede Subskription eine dynamische Warteschlange erstellt, werden diese Attributwerte aus der IBM WebSphere MQ classes for JMS-Modellwarteschlange SYSTEM.JMS.MODEL.QUEUE abgerufen. Um diese Einstellungen zu ändern, verwenden Sie Folgendes:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

Wenn der Rücksetzschwellenwert null ist, ist die Behandlung nicht verarbeitbarer Nachrichten inaktiviert und nicht verarbeitbare Nachrichten bleiben in der Eingabewarteschlange. Andernfalls wird die Nachricht an die namentlich genannte Rücksetzwarteschlange gesendet, sobald der Rücksetzungszähler den Schwellenwert erreicht. Wenn der Rücksetzungszähler den Schwellenwert erreicht, die Nachricht aber nicht in die Rücksetzwarteschlange eingereiht werden kann, wird die Nachricht an die Warteschlange für nicht zustellbare Nachrichten gesendet oder gelöscht. Diese Situation tritt ein, wenn die Rücksetzwarteschlange nicht definiert ist oder wenn das MessageConsumer-Objekt die Nachricht nicht an die Rücksetzwarteschlange senden kann. Weitere Informationen finden Sie im Abschnitt „Nachrichten aus der Warteschlange in ASF entfernen” auf Seite 987.

Wenn eine Nachricht wieder in die Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten eingereiht wird, ändern sich einige Feldwerte im Nachrichtendeskriptor (MQMD) der Nachricht. Weitere Informationen zum Format des MQMD finden Sie im Abschnitt MQMD - Nachrichtendeskriptor.

Wenn die Nachricht in die Rücksetzwarteschlange gelangt, ändern sich die Werte der folgenden MQMD-Felder.

- PutDate wird mit dem Datum aktualisiert, an dem die Nachricht in die Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten kommt.
- PutTime wird mit der Uhrzeit aktualisiert, zu der die Nachricht in die Warteschlange zum Wiedereinreihen zurückgesetzter Nachrichten kommt.
- Der Zurücksetzungszähler wird auf null zurückgesetzt.
- Der Ablauf der Nachricht wird aktualisiert, um die verbleibende Ablaufzeit zu dem Zeitpunkt widerzuspiegeln, an dem die ursprüngliche Nachricht von der JMS-Anwendung empfangen wurde.

Die Werte in den folgenden Feldern bleiben gleich, wenn die Nachricht in die Rücksetzwarteschlange kommt:

- StructId
- Version
- Bericht
- MessageType
- Feedback
- Encoding
- CodedCharSetId
- MsgId
- CorrelId
- ReplyToQ
- ReplyToQMgr
- Format
- Permanenz
- Priority

Ausnahmen in IBM WebSphere MQ classes for JMS

Eine Anwendung, die die IBM WebSphere MQ classes for JMS verwendet, muss in der Lage sein, die von JMS-API-Aufrufen ausgelösten oder an eine Ausnahmebehandlungsroutine übergebenen Ausnahmen zu handhaben.

IBM WebSphere MQ classes for JMS melden Laufzeitprobleme durch die Auslösung von Ausnahmebedingungen. JMSException ist die Stammklasse für von JMS-Methoden ausgelöste Ausnahmebedingungen und das Abfangen von JMSException-Ausnahmebedingungen bietet eine generische Art der Handhabung aller JMS-Ausnahmebedingungen.

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 oft das Ergebnis eines Problems auf untergeordneter Ebene, das von einer anderen mit dieser Ausnahmebedingung verlinkten Ausnahmebedingung gemeldet wurde. Eine Anwendung ruft eine verlinkte Ausnahmebedingung durch Aufruf der Methode JMSException.getLinkedException() oder Throwable.getCause() ab.

Bei den meisten der von IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS ausgelöst.
- Die nächste Ausnahmebedingung, eine Instanz von com.ibm.mq.MQException, wird durch den IBM WebSphere MQ-Messaging-Provider ausgelöst.
- Die nächste Ausnahmebedingung, eine Instanz von com.ibm.mq.jmqi.JmqiException, wird von der allgemeinen Java-Schnittstelle zur MQI ausgelöst.
- Die letzte Ausnahmebedingung, eine Instanz von java.net.ConnectionException, wird von der Java-Klassenbibliothek ausgelöst.

Weitere Informationen zu der Schichtarchitektur der IBM WebSphere MQ classes for JMS finden Sie unter „Architektur der IBM WebSphere MQ-Klassen für JMS” auf Seite 849.

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 WebSphere MQ-spezifische Informationen zu einem Problem abrufen

Instanzen von com.ibm.mq.MQException und com.ibm.mq.jmqi.JmqiException binden IBM WebSphere 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 948.

Upgrade von Vorgängerversionen von IBM WebSphere MQ classes for JMS durchführen

Im Vergleich zu früheren Versionen von IBM WebSphere MQ classes for JMShaben sich die meisten Fehlercodes und Ausnahmebedingungsnachrichten in Version 7 geändert. Der Grund für diese Änderungen ist, dass IBM WebSphere MQ classes for JMS jetzt über eine Schichtarchitektur verfügt und Ausnahmebedingungen von verschiedenen Ebenen im Code ausgelöst werden.

Versuchte eine Anwendung bei Vorgängerversionen von IBM WebSphere 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
```

In Version 7 der IBM WebSphere MQ classes for JMS wird in demselben Fall eine JMSException-Ausnahme mit den folgenden Informationen ausgelöst:

```
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: WebSphere 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 Ausnahmebedingungsnachrichten, die von der Methode Throwable.getMessage() zurückgegeben werden, oder Fehlercodes, die von der Methode JMSException.getErrorCode() zurückgegeben werden, von Ihrer Anwendung analysiert oder getestet werden und Sie ein Upgrade von einem älteren Release auf Version 7 vornehmen, müssen Sie die Anwendung wahrscheinlich ändern, damit sie Version 7 der IBM WebSphere MQ classes for JMS verwenden kann.

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 WebSphere 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.

V 7.5.0.8 Mit APAR IT14820 wurde ab IBM WebSphere MQ Version 7.5.0Fixpack 8 ein Fehler behoben, bei dem der JMS ExceptionListener einer Anwendung aufgrund von Ausnahmebedingungen aufgrund von Verbindungsunterbrechungen (z. B. MQRC_GET_INHIBITED) nicht aufgerufen wurde, obwohl die von der Anwendung verwendete Eigenschaft ASYNC_EXCEPTIONS der JMS-Verbindungsfactory auf ASYNC_EXCEPTIONS_ALL gesetzt war. Dies war vor Version 7.5.0, Fix Pack 8 der Standardwert.

V 7.5.0.8 Um das Verhalten aktueller JMS-Anwendungen beizubehalten, die einen JMS-MessageListener und einen JMS-ExceptionListener konfigurieren, und um sicherzustellen, dass die IBM WebSphere MQ classes for JMS der JMS-Spezifikation entsprechen, wurde der Standardwert der Eigenschaft ASYNC_EXCEPTIONS in der JMS-Verbindungsfactory für die IBM WebSphere MQ classes for JMS in ASYNC_EXCEPTIONS_CONNECTIONBROKEN geändert. Deshalb werden standardmäßig nur Ausnahmebedingungen, die Fehlercodes für Verbindungsunterbrechungen entsprechen, an den JMS ExceptionListener einer Anwendung übergeben.

V 7.5.0.8 Ab Version 7.5.0, Fix Pack 8 wurden außerdem die IBM WebSphere MQ classes for JMS dahingehend aktualisiert, dass JMSExceptions, die sich auf Fehler aufgrund von Verbindungsunterbrechungen beziehen und während der Nachrichtenübermittlung an asynchrone Nachrichtenkonsumenten auftreten, weiterhin an einen registrierten ExceptionListener übergeben werden, wenn die Eigenschaft ASYNC_EXCEPTIONS der von der Anwendung verwendeten JMS ConnectionFactory auf den Wert ASYNC_EXCEPTIONS_ALL gesetzt ist.

V 7.5.0.8 Weitere Informationen zu den Änderungen, die an Listenern für Ausnahmebedingungen für Version 7.5.0, Fix Pack 8 vorgenommen wurden, und zu den Gründen, warum die Änderungen gegenüber früheren Releases vorgenommen wurden, finden Sie unter JMS: Exception listener changes in Version 7.5.

Bei einem Fehler eines anderen Typs wird eine JMSException-Ausnahmebedingung durch den aktuellen JMS-API-Aufruf ausgelöst.

Wenn eine Anwendung keinen Listener für Ausnahmebedingungen im Connection-Objekt registriert, werden alle Ausnahmen, die an den Listener für Ausnahmebedingungen übermittelt worden wären, in das Protokoll der IBM WebSphere MQ classes for JMS geschrieben.

Zugehörige Verweise
ASYNCEXCEPTION

Protokollierung von Fehlern in WebSphere MQ Classes for JMS

Informationen zu Laufzeitproblemen, die möglicherweise eine Problembehebung durch den Benutzer erfordern, werden in das Protokoll von WebSphere MQ Classes for JMS geschrieben.

Wenn eine Anwendung beispielsweise versucht, eine Eigenschaft einer Verbindungsfactory festzulegen, der Name der Eigenschaft jedoch nicht erkannt wird, schreibt WebSphere MQ Classes for JMS Informationen zu dem Problem in das eigene Protokoll.

Standardmäßig hat die Datei mit dem Protokoll den Namen 'mqjms.log' und befindet sich im aktuellen Arbeitsverzeichnis. Sie können den Namen und die Position der Protokolldatei jedoch ändern, indem Sie die Eigenschaft 'com.ibm.msg.client.commonservices.log.outputName' in der Konfigurationsdatei von WebSphere MQ Classes for JMS entsprechend festlegen. Sie finden Informationen zur Konfigurationsdatei von WebSphere MQ Classes for JMS im Abschnitt „Die Konfigurationsdatei IBM WebSphere MQ classes for JMS” auf Seite 771. Der Abschnitt „Protokollierung und IBM WebSphere MQ classes for JMS” auf Seite 847 enthält weitere Details zu gültigen Werten für die Eigenschaft 'com.ibm.msg.client.commonservices.log.outputName'.

First Failure Support Technology (FFST) in den WebSphere MQ-Klassen für JMS

Wenn bei den WebSphere MQ-Klassen für JMS ein schwerwiegender interner Fehler auftritt, werden FFST-Informationen (FFST = First Failure Support Technology) generiert.

Die FFST-Informationen werden in eine Datei mit dem Namen JMSC*nnnn*.FDC geschrieben, wobei *nnnn* eine vierstellige Zahl ist. Diese Datei befindet sich in einem Verzeichnis mit dem Namen FFDC, das

ein Unterverzeichnis des Verzeichnisses ist, in das die Traceausgabe geschrieben wird. Standardmäßig wird die Traceausgabe in das aktuelle Arbeitsverzeichnis geschrieben, aber Sie können die Traceausgabe in ein anderes Verzeichnis umleiten, indem Sie die Eigenschaft **com.ibm.msg.client.commonservices.trace.outputName** in der Konfigurationsdatei von WebSphere MQ Classes for JMS festlegen. Informationen zur Konfigurationsdatei für WebSphere MQ Classes for JMS finden Sie im Abschnitt „Die Konfigurationsdatei IBM WebSphere MQ classes for JMS" auf Seite 771.

Wenn das Tracing aktiviert ist, während die FFST-Informationen generiert werden, werden die FFST-Informationen auch in die Tracedatei geschrieben. Weitere Informationen zur Traceerstellung für JMS-Programme finden Sie unter Trace für Anwendungen erstellen, die die IBM WebSphere MQ classes for JMS verwenden.

Um die Erstellung von FFDC-Dateien zu unterdrücken, legen Sie die Eigenschaft **com.ibm.msg.client.commonservices.ffst.suppress**wie folgt fest:

0
Alle FFDC-Dateien ausgeben (Standardeinstellung).

-1
Nur die ersten FFDC-Dateien eines bestimmten Typs ausgeben.

Ganzzahl
Alle FFDC-Dateien außer denen, die ein Vielfaches dieser Zahl sind, unterdrücken.

In einer WebSphere MQ Classes for JMS-Anwendung auf WebSphere MQ-Funktionen zugreifen

WebSphere MQ Classes for JMS stellt Einrichtungen bereit, mit denen mehrere Funktionen von WebSphere 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 in einer WebSphere MQ Classes for JMS-Anwendung 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 WebSphere MQ-Anwendungen müssen im MQMD der an sie gesendeten Nachrichten bestimmte Werte festgelegt sein. WebSphere MQ Classes for JMS stellt Nachrichtenattribute zur Verfügung, mit denen JMS-Anwendungen MQMD-Felder festlegen und somit WebSphere MQ-Anwendungen "steuern" können.

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 955.

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-Destination-Objekten
Zwei Eigenschaften des Destination-Objekts steuern den Zugriff auf den MQMD über JMS, während eine dritte Eigenschaft den Nachrichtenkontext steuert.

Tabelle 124. 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	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 125. 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. • YES Die JMS_IBM_MQMD*-Eigenschaften werden verarbeitet. Ihre Werte werden in die zugrunde liegende MQMD-Struktur kopiert.	• **False** • True	setMQMDWriteEnabled
WMQ_MQMD_READ_ENABLED	• **NO** Beim Senden von Nachrichten werden die JMS_IBM_MQMD*-Eigenschaften einer gesendeten Nachricht nicht mit den aktualisierten Feldwerten in der MQMD-Struktur aktualisiert. Beim Empfangen von Nachrichten ist keine der JMS_IBM_MQMD*-Eigenschaften in einer empfangenen Nachricht verfügbar, auch wenn der Absender einige oder alle dieser Eigenschaften festgelegt hatte. • YES 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 125. 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 126. Eigenschaftsnamen, Beschreibungen und Typen

Eigenschaft	Beschreibung	Java-Typ	Link zur ausführlichen Beschreibung
JMS_IBM_MQMD_Report	Optionen für Berichtsnachrichten	Integer	Bericht
JMS_IBM_MQMD_MsgType	Nachrichtentyp	Integer	MsgType
JMS_IBM_MQMD_Expiry	Nachrichtenlebensdauer	Integer	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	Priorität
JMS_IBM_MQMD_Persistence	Nachrichtenpersistenz	Integer	Permanenz
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 126. 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

Tabelle 126. Eigenschaftsnamen, Beschreibungen und Typen (Forts.)

Eigenschaft	Beschreibung	Java-Typ	Link zur ausführlichen Beschreibung

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, und 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 in einer Nachricht verstößt gegen die JMS-Spezifikation.

Zugriff auf IBM WebSphere MQ Nachrichtendaten über eine Anwendung, die die WebSphere MQ-Klassen für JMS verwendet

Sie können mit IBM WebSphere MQ -Klassen für JMS auf die vollständigen WebSphere MQ -Nachrichtendaten in einer Anwendung 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 WebSphere 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 WebSphere-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, generiert WebSphere 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, sendet WebSphere 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 WebSphere MQ-Nachricht 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 WebSphere 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 WebSphere 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 WebSphere 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 gelten für den IBM WebSphere MQ-JMS-Provider.
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 163 auf Seite 959 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 "ëÈÊ'>...??>?"
   ```

Abbildung 163. Inkonsistente Codierung von MQMD und Nachrichtendaten

Beide Codeausschnitte in Abbildung 164 auf Seite 960 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 164. Senden einer Nachricht mit einem MQ-Nachrichtenhauptteil

Nachricht empfangen

Wenn WMQ_MESSAGE_BODY auf WMQ_MESSAGE_BODY_JMS gesetzt ist, werden 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 JMSBytesMessage der Typ der eingehenden JMS-Nachricht. 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.

Ist WMQ_MESSAGE_BODY auf den Standardwert WMQ_MESSAGE_BODY_UNSPECIFIED gesetzt, wird dieser von den IBM WebSphere MQ-Klassen für JMS auf WMQ_MESSAGE_BODY_JMS gesetzt.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ-Nachricht als Teil der Nachrichtennutzdaten verarbeitet (also als Teil des JMS-Nachrichtenhauptteils).

Tabelle 127. Eigenschaftsnamen und Beschreibungen

Eigenschaft	Kurzform	Beschreibung
WMQ_MESSAGE_BODY	MBODY	Bestimmt, ob eine JMS-Anwendung den MQRFH2 einer WebSphere MQ-Nachricht als Teil der Nachrichtennutzdaten verarbeitet (also als Teil des JMS-Nachrichtenhauptteils).

Tabelle 128. Eigenschaftsnamen, Werte und Festlegungsmethoden

Eigenschaft	Gültige Werte im Verwaltungstool (Standardwerte sind fett dargestellt)	Gültige Werte in Programmen	Festlegungsmethode
WMQ_MESSAGE_BODY	• **UNSPECIFIED** Beim Senden generiert WebSphere 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 WebSphere MQ Classes for JMS automatisch einen MQRFH2-Header und fügt ihn in die WebSphere MQ-Nachricht ein. Beim Empfang legt WebSphere MQ Classes for JMS die JMS-Nachrichteneigenschaften in Übereinstimmung mit den Werten im MQRFH2 fest (falls vorhanden); der MQRFH2 wird nicht als Teil des JMS-Nachrichtenhauptteils dargestellt. • MQ Beim Senden generiert WebSphere MQ Classes for JMS keinen MQRFH2-Header. Beim Empfang stellt WebSphere MQ Classes for JMS den MQRFH2 als Teil des JMS-Nachrichtenhauptteils dar.	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

Persistente JMS-Nachrichten

WebSphere MQ Classes for JMS-Anwendungen bieten mit dem Warteschlangenattribut **NonPersistentMessageClass** bessere Leistung für persistente JMS-Nachrichten, jedoch zu Lasten der Zuverlässigkeit.

Eine WebSphere MQ-Warteschlange weist das Attribut **NonPersistentMessageClass** auf. Der Wert dieses Attributs bestimmt, ob nicht persistente Nachrichten in der Warteschlange beim Neustart des Warteschlangenmanagers gelöscht werden.

Dieses Attribut können Sie für eine lokale Warteschlange mit dem WebSphere MQ Script-Befehl (MQSC-Befehl) DEFINE QLOCAL mit einem der folgenden Parameter festlegen:

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 die Leistung persistenter JMS-Nachrichten in WebSphere MQ Classes for JMS-Anwendungen mithilfe dieses Warteschlangenattributs verbessert wird.

Die Eigenschaft PERSISTENCE eines Queue- oder Topic-Objekts kann den Wert HIGH haben. Diesen Wert können Sie mit dem WebSphere 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 oder eine nicht persistente JMS-Nachricht an ein Ziel sendet, dessen Eigenschaft PERSISTENCE den Wert HIGH hat, die zugehörige WebSphere MQ-Warteschlange aber auf NPMCLASS(HIGH) gesetzt ist, wird die Nachricht als nicht persistente WebSphere MQ-Nachricht in die Warteschlange eingereiht. Weist die Eigenschaft PERSISTENCE des Ziels hingegen nicht den Wert HIGH auf bzw. ist die zugehörige Warteschlange auf NPMCLASS(NORMAL) gesetzt, so wird eine persistente JMS-Nachricht als persistente WebSphere MQ-Nachricht und eine nicht persistente JMS-Nachricht als nicht persistente WebSphere MQ-Nachricht in die Warteschlange eingereiht.

Wenn eine persistente JMS-Nachricht als nicht persistente WebSphere MQ-Nachricht in eine Warteschlange eingereiht wird und Sie sicherstellen möchten, dass die Nachricht nach einer kontrollierten oder sofortigen Beendigung eines Warteschlangenmanagers nicht verworfen wird, müssen alle Warteschlangen, über die die Nachricht weitergeleitet werden kann, auf NPMCLASS(HIGH) gesetzt sein. In der Publish/Subscribe-Domäne gehören zu diesen Warteschlangen auch Warteschlangen für Subskribenten. Als Unterstützung bei der Umsetzung dieser Konfiguration löst WebSphere 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 WebSphere 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 Nachricht, die als persistente JMS-Nachricht gesendet wird, wird als persistente JMS-Nachricht empfangen, und eine Nachricht, die als nicht persistente JMS-Nachricht gesendet wird, wird als nicht persistente JMS-Nachricht empfangen.

Wenn eine Anwendung die erste Nachricht an ein Ziel sendet oder den ersten Nachrichtenkonsumenten für ein Ziel erstellt, dessen Eigenschaft PERSISTENCE auf HIGH gesetzt ist, gibt WebSphere MQ Classes for JMS einen MQINQ-Aufruf aus, um festzustellen, ob für die zugehörige WebSphere MQ-Warteschlange NPMCLASS(HIGH) gesetzt ist. Daher muss die Anwendung zur Abfrage der Warteschlange berechtigt sein. Außerdem speichert WebSphere MQ Classes for JMS das Ergebnis des MQINQ-Aufrufs, bis das Ziel gelöscht wird. Es werden keine weiteren MQINQ-Aufrufe ausgegeben. Dies bedeutet aber, dass WebSphere MQ Classes for JMS eine Änderung der NPMCLASS-Einstellung der zugehörigen Warteschlange nicht bemerken würde, so lange die Anwendung dieses Ziel verwendet.

Wenn Sie die Einreihung persistenter JMS-Nachrichten in WebSphere MQ-Warteschlangen als nicht persistente WebSphere MQ-Nachrichten erlauben, erzielen Sie zwar eine bessere Leistung, jedoch zu Lasten der Zuverlässigkeit. Ist für Sie Zuverlässigkeit wichtiger als Leistung, dürfen Sie also persistente JMS-Nachrichten nicht an ein Ziel senden, dessen PERSISTENCE-Eigenschaft auf HIGH gesetzt ist.

Die JMS-Schicht kann auch SYSTEM.JMS.TEMPQ.MODEL statt 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.

Secure Sockets Layer (SSL) mit WebSphere MQ Classes for JMS verwenden

WebSphere MQ Classes for JMS-Anwendungen können die SSL-Verschlüsselung verwenden. Hierfür benötigen sie einen JSEE-Provider.

WebSphere MQ Classes for JMS-Verbindungen, die TRANSPORT(CLIENT) verwenden, unterstützen die SSL-Verschlüsselung (Secure Sockets Layer). SSL stellt eine Kommunikationsverschlüsselung, Authentifizierung und Nachrichtenintegrität bereit. In der Regel wird damit die Kommunikation zwischen zwei Peers im Internet oder innerhalb eines Intranets geschützt.

WebSphere MQ Classes for JMS verwendet Java Secure Socket Extension (JSSE) für die SSL-Verschlüsselung und erfordert 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 WebSphere MQ Classes for JMS-Anwendung für die Verbindung mit einem Warteschlangenmanager eine Definitionstabelle für Clientkanäle (CCDT) verwendet, lesen Sie den Abschnitt „Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for JMS” auf Seite 972.

Objekteigenschaft SSLCIPHERSUITE
Legen Sie SSLCIPHERSUITE fest, um die SSL-Verschlüsselung bei einem ConnectionFactory-Objekt zu aktivieren.

Um die SSL-Verschlüsselung bei einem ConnectionFactory-Objekt zu aktivieren, verwenden Sie JMSAdmin, um die Eigenschaft SSLCIPHERSUITE auf eine Cipher-Suite festzulegen, die durch Ihren JSSE-Provider unterstützt wird. Dieser Wert muss mit der für den Zielkanal festgelegten CipherSpec übereinstimmen. Da sich Cipher-Suites jedoch von CipherSpecs unterscheiden, haben sie unterschiedliche Namen. „SSL CipherSpecs und Cipher-Suites in JMS” auf Seite 967 enthält eine Tabellenzuordnung zwischen den von WebSphere MQ unterstützten CipherSpecs und ihren Cipher-Suite-Entsprechungen, die in JSSE bekannt sind. Weitere Informationen zu CipherSpecs und Cipher-Suites mit WebSphere MQ finden Sie in Sicherheit.

Wenn Sie beispielsweise ein ConnectionFactory-Objekt einrichten möchten, das für die Erstellung einer Verbindung über einen SSL-fähigen MQI-Kanal mit dem CipherSpec-Wert RC4_MD5_EXPORT verwendet werden kann, geben Sie folgenden Befehl an JMSAdmin aus:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

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 Clientkanäle mit IBM WebSphere MQ classes for JMS” auf Seite 972.

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

Der Standardwert dieser Eigenschaft lautet NO, d. h., eine Verbindung kann jede Cipher-Suite verwenden, die durch WebSphere MQ unterstützt wird.

Wenn eine Anwendung mehrere Verbindungen verwendet, bestimmt der Wert von SSLFIPSREQUIRED, der bei der ersten Erstellung der Verbindung durch die Anwendung verwendet wird, den Wert, der verwendet wird, wenn die Anwendung nachfolgende Verbindungen erstellt. Dies bedeutet, dass der Wert der Eigenschaft SSLFIPSREQUIRED der Verbindungsfactory, der für die Erstellung einer nachfolgenden Verbindung verwendet wird, ignoriert wird. Sie müssen die Anwendung erneut starten, wenn Sie einen anderen Wert für SSLFIPSREQUIRED verwenden möchten.

Eine Anwendung kann diese Eigenschaft festlegen, indem sie die Methode setSSLFipsRequired() eines ConnectionFactory-Objekts aufruft. Die Eigenschaft wird ignoriert, wenn keine Cipher-Suite festgelegt wurde.

Zugehörige Tasks
Angeben, dass nur FIPS-zertifizierte CipherSpecs während der Ausführung auf dem MQI-Client verwendet werden
Zugehörige Verweise
Federal Information Processing Standards (FIPS) für UNIX, Linux und 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 ein ConnectionFactory-Objekt festgelegt, bei dem erwartet wird, dass sich der Warteschlangenmanager mit einem allgemeinen Namen identifiziert, der mit den Zeichen `QMGR.` beginnt. Außerdem müssen mindestens zwei Namen für die Organisationseinheit vorhanden sein, wovon der erste IBM und der zweite WEBSPHERE lauten 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. Das folgende JMSAdmin-Beispiel weist JMS an, eine CRL 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: Wenn Sie einen CertStore erfolgreich mit einer CRL verwenden möchten, die auf einem LDAP-Server gehostet wird, stellen Sie sicher, 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. Diese Anzahl an Bytes schließt auch Steuerinformationen ein, die von WebSphere MQ Classes for JMS gesendet und empfangen wurden.

Beispiel: Um ein ConnectionFactory-Objekt zu konfigurieren, das zum Herstellen einer Verbindung über einen für SSL aktivierten MQI-Kanal mit einem geheimen Schlüssel, der nach 4 MB gesendeten und empfangenen Daten erneut vereinbart wird, verwendet werden kann, müssen Sie den folgenden Befehl an JMSAdmin ausgeben:

```
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 SSL-Verbindung für eine Anwendung anpassen möchten, erstellen Sie eine SSLSocketFactory und konfigurieren Sie JMS für ihre Verwendung.

Möglicherweise möchten Sie weitere Aspekte der SSL-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 WebSphere MQ Classes for JMS-Anwendungen 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.

SSL CipherSpecs und Cipher-Suites in JMS
Von WebSphere MQ unterstützte CipherSpecs und die entsprechenden Cipher-Suites.

In Tabelle 129 auf Seite 967 sind die von WebSphere MQ unterstützten CipherSpecs und die entsprechenden Cipher-Suites aufgeführt. Wenn die ConnectionFactory-Eigenschaft SSLFIPSREQUIRED auf den Wert NO gesetzt ist, kann eine Anwendung, die die WebSphere MQ-Klassen für JMS verwendet, eine Verbindung zu einem Warteschlangenmanager herstellen, wenn eine unterstützte CipherSpec am Serverende des MQI-Kanals und die funktional entsprechende Cipher-Suite am Clientende angegeben ist. Wenn SSLFIPSREQUIRED auf YES gesetzt ist, bestimmt die Kombination aus CipherSpec und Cipher-Suite, ob die Anwendung eine Verbindung zum Warteschlangenmanager herstellen kann.

Am Serverende eines MQI-Kanals kann der Name einer CipherSpec als Wert des Parameters "SSLCIPH" in einem Befehl "DEFINE CHANNEL CHLTYPE(SVRCONN)" angegeben werden. Am Clientende eines MQI-Kanals kann der Name einer Cipher-Suite wie folgt angegeben werden:

- Eine Anwendung kann die Methode setSSLCipherSuite() eines ConnectionFactory-Objekts aufrufen.
- Mit dem WebSphere MQ-JMS-Verwaltungstool können Sie die Eigenschaft SSLCIPHERSUITE eines ConnectionFactory-Objekts festlegen.

Tabelle 129. Von WebSphere MQ unterstützte CipherSpecs und die entsprechenden Cipher-Suites

CipherSpec	**Entsprechende Cipher-Suite**	**Verbindung möglich, wenn SFIPS [1] auf YES gesetzt ist?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	Nein
NULL_SHA	SSL_RSA_WITH_NULL_SHA	Nein
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	Nein
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	Nein
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	Nein
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	Nein
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	Nein
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	Nein
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	Nein
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Nein
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	Nein [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	Ja [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	Ja [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	Ja [5] [7]
AES_SHA_US [2]		
TLS_RSA_WITH_DES_CBC_SHA [8] [9]	SSL_RSA_WITH_DES_CBC_SHA	Nein [3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA [8]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Ja
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	Nein [4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	Nein [6]

Anmerkungen:

1. Wenn das WebSphere MQ-JMS-Verwaltungstool verwendet wird, ist SFIPS der Kurzname der ConnectionFactory-Eigenschaft SSLFIPSREQUIRED.
2. Für diese CipherSpec ist keine funktional entsprechende Cipher-Suite vorhanden.
3. Diese CipherSpec wurde vor dem 19. Mai 2007 nach FIPS 140-2 zertifiziert.
4. Diese CipherSpec wurde vor dem 19. Mai 2007 nach FIPS 140-2 zertifiziert. Der Name FIPS_WITH_DES_CBC_SHA ist historisch und gibt die Tatsache wieder, dass diese Verschlüsselungsspezifikation (CipherSpec) zuvor mit FIPS konform war. Dies ist jedoch nicht mehr der Fall. Diese CipherSpec ist veraltet und sollte nicht mehr verwendet werden.
5. Verbindungen zwischen WebSphere MQ Explorer und einem Warteschlangenmanager können mit diesen CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) nur gesichert werden, wenn für die von WebSphere MQ Explorer verwendete JRE die uneingeschränkten Richtliniendateien verwendet werden.

   Weitere Informationen zu Richtliniendateien finden Sie in den Sicherheitsinformationen.
6. Der Name FIPS_WITH_3DES_EDE_CBC_SHA ist historisch und gibt die Tatsache wieder, dass diese Verschlüsselungsspezifikation (CipherSpec) zuvor mit FIPS konform war. Dies ist jedoch nicht mehr der Fall. Diese CipherSpec ist veraltet und sollte nicht mehr verwendet werden.
7. Für diese CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) sind die IBM JREs 6.0 SR13 FP2, 7.0 SR4 FP2 oder höher erforderlich.
8. Für diese CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) kann SSLv3 oder TLS verwendet werden. Wenn FIPS nicht aktiviert ist, wird standardmäßig SSLv3 verwendet. Wenn TLS verwendet werden soll, muss die Java-Systemeigenschaft **com.ibm.mq.cfg.preferTLS** auf `true` gesetzt werden.
9. Die CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA wird nicht weiter unterstützt. Nach wie vor sind mit dieser CipherSpec jedoch noch Datenübertragungen bis zu 32 GB möglich, bevor die Verbindung mit Fehler AMQ9288 beendet wird. Zur Vermeidung dieses Fehlers sollten Sie entweder auf Triple DES verzichten oder, wenn Sie diese CipherSpec verwenden möchten, die Zurücksetzung von geheimen Schlüsseln aktivieren.

Zugehörige 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

Kanalexits in Java für WebSphere MQ Classes for JMS schreiben

Sie erstellen Kanalexits, indem Sie Java-Klassen definieren, 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 von WebSphere 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 SSL-Verschlüsselung *nach* dem Aufruf der Kanalexits. Analog hierzu erfolgt die SSL-Entschlüsselung beim Datenempfang von der Serverseite einer Verbindung *vor* dem Aufruf der Kanalexits.

In früheren Versionen von WebSphere MQ Classes for JMS als Version 7.0 wurden Kanalexits über die 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 WebSphere MQ classes for JMS für Verwendung von Kanalexits konfigurieren

Eine Anwendung der IBM WebSphere 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 WebSphere MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **sendexit** in einem ConnectionFactory-Objekt, die vom IBM WebSphere 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 WebSphere MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **receiveexit** in einem ConnectionFactory-Objekt, die vom IBM WebSphere 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 WebSphere MQ-Ressourcenadapter für die eingehende Kommunikation verwendet wird.
- Die Eigenschaft **securityexit** in einem ConnectionFactory-Objekt, die vom IBM WebSphere MQ-Ressourcenadapter für die abgehende Kommunikation verwendet wird.

Für MQConnectionFactories können Sie die Eigenschaften **SENDEXIT**, **RECEXIT** und **SECEXIT** mit dem IBM WebSphere MQ JMS-Verwaltungstool oder dem IBM WebSphere MQ Explorer festlegen. Alternativ kann eine Anwendung die Eigenschaften durch Aufrufe der Methoden `setSendExit()`, `setReceiveExit()` und `setSecurityExit()` festlegen.

Kanalexits werden durch ihr eigenes Klassenladeprogramm geladen. Um einen Kanalexit zu finden, durchsucht das Klassenladeprogramm die folgenden Positionen in der angegebenen Reihenfolge.

1. Der Klassenpfad, der durch die Eigenschaft **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** oder durch das Attribut **JavaExitsClassPath** in der Zeilengruppe 'Channels' der IBM WebSphere 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 WebSphere MQ-Exitverzeichnis wie in Tabelle 130 auf Seite 971 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 130. Das IBM WebSphere MQ-Verzeichnis 'exits'

Plattform	**Directory**
UNIX and Linux	`/var/mqm/exits` (32-Bit-Kanalexits) `/var/mqm/exits64` (64-Bit-Kanalexits)
Windows	*Installationsdatenverzeichnis* \exits Dabei steht *Installationsdatenverzeichnis* für das Verzeichnis, das Sie während der Installation für die IBM WebSphere MQ-Datendateien ausgewählt haben. Das Standardverzeichnis ist `C:\Program Files\IBM\WebSphere MQ`.

Anmerkung: Wenn ein Kanalexit an mehr als einer Position vorhanden ist, laden die IBM WebSphere MQ classes for JMS die erste Instanz, die sie finden.

Dem Klassenladeprogramm ist das Klassenladeprogramm übergeordnet, mit dem die IBM WebSphere 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 WebSphere 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.

Bei Ausführung der Anwendung mit aktiviertem Java Security Manager muss die Richtlinienkonfigurationsdatei, die von der JRE (Java), in der die Anwendung aktiv ist, 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 Versionen von IBM WebSphere MQ vor Version 7.0 geliefert wurden, werden weiterhin unterstützt. Bei Verwendung von Kanalexits, die diese Schnittstellen implementieren, muss `com.ibm.mq.jar` im Klassenpfad enthalten sein.

Informationen zum Schreiben von Kanalexits in der Programmiersprache C finden Sie unter „Kanalexitprogramme für Messaging-Kanäle” auf Seite 422. Sie müssen Kanalexitprogramme, die in C oder C++ geschrieben wurden, in dem Verzeichnis speichern, das in Tabelle 130 auf Seite 971 aufgelistet wird.

Wenn Ihre Anwendung für die Verbindung zu einem Warteschlangenmanager eine Definitionstabelle für Clientkanäle (CCDT) verwendet, lesen Sie den Abschnitt „Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for JMS” auf Seite 972.

Bei Verwendung von WebSphere 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 ge-

trennt 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 WebSphere MQ-JMS-Verwaltungstool oder mit WebSphere 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 WebSphere MQ-JMS-Verwaltungstool oder mit WebSphere 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 Clientkanäle mit IBM WebSphere MQ classes for JMS” auf Seite 972.

Definitionstabelle für Clientkanäle mit IBM WebSphere MQ classes for JMS

Eine Anwendung, die die IBM WebSphere MQ-Klassen für JMS verwendet, kann Definitionen von Clientverbindungskanälen verwenden, die in einer Definitionstabelle für Clientkanäle (CCDT) gespeichert sind. Sie konfigurieren ein ConnectionFactory-Objekt, um die CCDT zu verwenden. Für die Verwendung gelten einige Einschränkungen.

Alternativ zum Erstellen einer Clientverbindungskanaldefinition durch Festlegen bestimmter Eigenschaften eines ConnectionFactory -Objekts kann eine IBM WebSphere MQ classes for JMS -Anwendung Clientverbindungskanaldefinitionen verwenden, die in einer Clientkanaldefinitionstabelle gespeichert sind. Diese Definitionen werden mit IBM WebSphere MQ-Scriptbefehlen (MQSC) oder IBM WebSphere MQ-PCF-Befehlen (Programmable Command Format) erstellt. Wenn die Anwendung ein Verbindungsobjekt erstellt, durchsucht IBM WebSphere MQ classes for JMS die Definitionstabelle für Clientkanäle nach einer geeigneten Definition für Clientverbindungskanäle und verwendet die Kanaldefinition zum Starten 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. 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 JMS-Verwaltungstool von IBM WebSphere MQ setzen; ebenso kann die Eigenschaft von einer Anwendung gesetzt 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
- Ein Stern (*)
- Eine leere Zeichenfolge oder eine Zeichenfolge, die nur Leerzeichen enthält

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 JMS-Verwaltungstool von WebSphere MQ oder mit IBM WebSphere MQ Explorer setzen. Alternativ kann eine Anwendung die Eigenschaft angeben, indem sie die Methode setQueueManager() des ConnectionFactory-Objekts aufruft.

Wenn eine Anwendung dann ein Connection-Objekt aus dem Objekt ConnectionFactory erstellt, greift IBM WebSphere MQ classes for JMS auf die Clientkanaldefinitionstabelle zu, die durch die Eigenschaft CCDTURL angegeben ist, verwendet die Eigenschaft QMANAGER, um die Tabelle nach einer geeigneten Clientverbindungskanaldefinition zu durchsuchen, und verwendet dann die Kanaldefinition, um einen MQI-Kanal zu einem Warteschlangenmanager zu starten.

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 WebSphere MQ classes for JMS eine geeignete Clientverbindungskanaldefinition in der Definitionstabelle für Clientkanäle findet, verwendet es nur die aus der Tabelle extrahierten Informationen zum Starten eines MQI-Kanals. Alle kanalbezogenen Eigenschaften des ConnectionFactory-Objekts werden ignoriert.

Beachten Sie inbesondere die folgenden Punkte, wenn Sie Secure Sockets Layer (SSL) verwenden:

- Ein MQI-Kanal verwendet SSL nur, wenn die aus der Clientkanaldefinitionstabelle extrahierte Kanaldefinition den Namen einer CipherSpec angibt, die von IBM WebSphere MQ classes for JMSunterstützt 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 WebSphere MQ classes for JMS verwendet nur diese Informationen für den Zugriff auf LDAP-Server, die CRLs enthalten.
- Eine Definitionstabelle für Clientkanäle kann auch die Position eines OCSP-Responders (Online Certificate Status Protocol) enthalten. IBM WebSphere MQ classes for JMS kann die OCSP-Informationen in einer Clientkanaldefinitionstabellendatei nicht verwenden. Allerdings können Sie OCSP gemäß Abschnitt Online Certificate Protocol verwenden konfigurieren.

Weitere Informationen zur Verwendung von SSL in Verbindung mit einer Definitionstabelle für den Clientkanal finden Sie unter Erweiterten transaktionsorientierten Client mit SSL-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 Clientkanäle extrahiert wurde, kann in Java geschriebene Kanalexits angeben. 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 WebSphere MQ Classes for JMS schreiben” auf Seite 968.

  Die Verwendung von Kanalexits, die in einer anderen Sprache als Java geschrieben sind, 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 Wiederherstellung der JMS-Clientverbindung

Konfigurieren Sie Ihren JMS-Client so, dass er die Verbindung nach einem Netz-, Warteschlangenmanager- oder Serverfehler automatisch wiederholt.

Verwenden Sie die Eigenschaften `CONNECTIONNAMELIST` und `CLIENTRECONNECTOPTIONS` der Klasse `MQConnectionFactory`, um eine Clientverbindung so zu konfigurieren, dass bei einem Verbindungsfehler oder einer Verwaltungsanforderung zum Wiederherstellen der Verbindung von Clientanwendungen nach dem Stoppen eines Warteschlangenmanagers die Verbindung automatisch wiederhergestellt wird.

Die vollständige Liste der Verbindungsnamen in einer connectionNameList ist nur den set/getconnectionNameList-Methoden zugänglich, die eine Liste von Verbindungsnamen handhaben können. Methoden wie get/setHostname, die keine Namenslisten handhaben, greifen auf den ersten Namen in der Liste zu.

Automatisch wiederherstellbare Clientverbindungen werden erst dann wiederherstellbar, wenn die Verbindung hergestellt worden ist.

Ob eine Anwendung nach der automatischen Wiederherstellung der Verbindung weiterhin ordnungsgemäß funktioniert, hängt von ihrer Gestaltung ab. Informieren Sie sich in den entsprechenden Abschnitten, wie Clients, deren Verbindung wiederhergestellt werden kann, gestaltet werden müssen. Einige vorhandene Clients funktionieren möglicherweise ohne Änderung nach einer automatischen Wiederherstellung der Verbindung ordnungsgemäß.

Die automatische Wiederherstellung einer Clientverbindung wird von WebSphere MQ Classes for Java nicht unterstützt.

Um zu verhindern, dass alle Clients, die mit einem ausgefallenen Warteschlangenmanager verbunden sind, gleichzeitig die Verbindung wiederherstellen, finden die Versuche zur Wiederherstellung der Verbindung um teils feste, teils zufällige Intervalle verzögert statt.

Standardmäßig erfolgen die Verbindungswiederholungsversuche in folgenden Intervallen:

1. Der erste Versuch nach einer Anfangsverzögerung von einer Sekunde plus einem zufälligen Element von bis zu 250 Millisekunden.
2. Der zweite Versuch zwei Sekunden, plus einem zufälligen Intervall von bis zu 500 Millisekunden, nachdem der erste Versuch fehlgeschlagen ist.
3. Der dritte Versuch vier Sekunden, plus einem zufälligen Intervall von bis zu einer Sekunde, nachdem der zweite Versuch fehlgeschlagen ist.
4. Der vierte Versuch acht Sekunden, plus einem zufälligen Intervall von bis zu zwei Sekunden, nachdem der dritte Versuch fehlgeschlagen ist.
5. Der fünfte Versuch 16 Sekunden, plus einem zufälligen Intervall von bis zu vier Sekunden, nachdem der vierte Versuch fehlgeschlagen ist.
6. Der sechste Versuch und alle nachfolgenden Versuche erfolgen 25 Sekunden plus einem zufälligen Intervall von bis zu sechs Sekunden und 250 Millisekunden, nachdem der vorherige Versuch fehlgeschlagen ist.

Dieser Prozess der Verbindungswiederholung wird fortgesetzt, bis die Verbindung vom Client zum Warteschlangenmanager wiederhergestellt ist oder das maximale Verbindungswiederholungsintervall verstrichen ist.

Wenn Sie die Standardwerte erhöhen müssen, um der erforderlichen Zeitspanne für die Wiederherstellung des Warteschlangenmanagers oder die Aktivierung des Standby-Warteschlangenmanagers besser Rechnung zu tragen, dann ändern Sie die Verzögerungswerte in der Datei MQCLIENT.INI mithilfe des Attributs **ReconDelay**.

Zugehörige Konzepte

Automatische Wiederherstellung der Clientverbindung

Zugehörige Tasks

Client mit einer Konfigurationsdatei konfigurieren

Gemeinsame Nutzung einer TCP/IP-Verbindung in IBM WebSphere MQ classes for JMS

Mehrere Instanzen eines MQI-Kanals können so festgelegt werden, dass sie eine einzelne TCP/IP-Verbindung gemeinsam nutzen.

Anwendungen, die in derselben Java Runtime Environment ausgeführt werden und die die IBM WebSphere MQ classes for JMS oder den IBM WebSphere MQ -Ressourcenadapter verwenden, um über den CLIENT-Transport eine Verbindung zu einem Warteschlangenmanager herzustellen, können für die gemeinsame Nutzung derselben Kanalinstanz eingerichtet werden.

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.

Beim Start einer Aktivierungsspezifikation startet der Ressourcenadapter der IBM WebSphere MQ-Klassen für JMS für die Aktivierungsspezifikation einen Datenaustausch 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 mit IBM WebSphere MQ classes for JMSverwendet wird, ist nicht garantiert, dass eine neue Verbindungsanforderung immer eine bereits eingerichtete Verbindung gemeinsam nutzt.

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 Kanalinstanzen = (<**maxPoolDepth**> + 1)/<**SHARECNV**>

Dabei ist <**maxPoolDepth**> der Wert der Eigenschaft **maxPoolDepth** und <**SHARECNV**> der Wert der Eigenschaft **SHARECNV** auf dem Kanal, der von der Aktivierungsspezifikation verwendet wird.

Andere JMS-Anwendungen

Anzahl der Kanalinstanzen = (< JMS-Verbindungen> + < JMS-Sitzungen>)/<**SHARECNV**>

Dabei ist < JMS-Verbindungen > die Anzahl der Verbindungen, die von der Anwendung erstellt werden, wobei < JMS-Sitzungen > die Anzahl der von der Anwendung erstellten JMS-Sitzungen und <**SHARECNV**> der Wert der Eigenschaft **SHARECNV** für den Kanal ist, 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 WebSphere MQ classes for JMS oder mit dem IBM WebSphere MQ classes for JMS-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.

Portbereich für Clientverbindungen in WebSphere 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 WebSphere MQ Classes for JMS-Anwendung versucht, im Clientmodus eine Verbindung zu einem WebSphere 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 WebSphere 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 Ausnahmebedingung des Typs 'JMSException' mit einer eingebetteten MQ-Ausnahmebedingung (MQException) ausgelöst, die den WebSphere MQ-Ursachencode MQRC_Q_MGR_NOT_AVAILABLE und die 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 WebSphere MQ Classes for JMS möglicherweise andere Verbindungen erstellt 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 WebSphere MQ Classes for JMS benötigt möglicherweise drei oder vier zusätzliche Ports. Wenn ein Verbindungsfehler auftritt, erhöhen Sie den Portbereich.

Das in WebSphere 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 WebSphere MQ Classes for JMS

Eine WebSphere MQ Classes for JMS-Anwendung kann WebSphere MQ-Funktionen verwenden, um einen Nachrichtenheader oder Daten zu komprimieren.

Durch die Komprimierung der durch einen WebSphere MQ-Kanal fließenden Daten können Sie die Leistung des Kanals verbessern und den Netzverkehr verringern. Mithilfe der in WebSphere 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 WebSphere MQ Classes for JMS-Anwendung 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 in IBM WebSphere MQ Classes for JMS asynchron 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 PUTSYNCALLOWED (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 WebSphere 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 131. PUTASYNCALLOWED- und DEFPRESP-Eigenschaften zur Bestimmung, ob Nachrichten asynchron eingereiht werden

WebSphere MQ-Warteschlangeneigenschaft	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST oder AS_Q_DEF oder AS_T_DEF**
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 allen nicht persistenten Nachrichten, die in einer nicht ausgeführten Sitzung gesendet werden, gibt die Eigenschaft SENDCHECKCOUNT des *ConnectionFactory*-Objekts an, wie viele Nachrichten gesendet

werden sollen, bevor IBM WebSphere MQ Classes for JMS prüft, ob der Warteschlangenmanager die Nachrichten tatsächlich 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, ruft IBM WebSphere 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 mit WebSphere MQ Classes for JMS verwenden

Die Vorauslesefunktion, die von WebSphere MQ bereitgestellt wird, ermöglicht es, dass nicht persistente Nachrichten, die außerhalb einer Transaktion empfangen werden, an IBM WebSphere MQ classes for JMS gesendet werden, bevor sie von einer Anwendung angefordert werden. Die IBM WebSphere 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 WebSphere 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 Topic, die bzw. das vom JMS-Ziel dargestellt wird, diese Option erlaubt.
- IBM WebSphere 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 132. Eigenschaften READAHEADALLOWED und DEFREADA zur Festlegung, ob das Vorauslesen beim Abrufen oder Durchsuchen von nicht persistenten Nachrichten außerhalb einer Transaktion verwendet wird

WebSphere MQ-Zieleigenschaft	**READAHEADALLOWED= YES**	**READAHEADALLOWED= NO**	**AS_DEST oder AS_Q_DEF or AS_T_DEF**
WebSphere MQ-Warteschlangeneigenschaft			
DEFREADA = NO	Vorauslesefunktion aktiviert	Vorauslesefunktion nicht aktiviert	Vorauslesefunktion nicht aktiviert

Tabelle 132. Eigenschaften READAHEADALLOWED und DEFREADA zur Festlegung, ob das Vorauslesen beim Abrufen oder Durchsuchen von nicht persistenten Nachrichten außerhalb einer Transaktion verwendet wird (Forts.)

WebSphere MQ-Zieleigenschaft	READAHEADALLOWED= YES	READAHEADALLOWED= NO	AS_DEST oder AS_Q_DEF or AS_T_DEF
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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS die nächste Nachricht im internen Puffer zurück.

Die IBM WebSphere MQ classes for JMS fordern weitere nicht persistente Nachrichten vom Warteschlangenmanager an, wenn der interne Puffer leer ist.

Der interne Puffer, der vom IBM WebSphere MQ classes for JMS verwendet wird, wird gelöscht, wenn eine Anwendung eine `MessageConsumer`oder die JMS-Sitzung schließt, der eine `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 WebSphere MQ Classes for JMS

Ein WebSphere MQ Classes for JMS-Client 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. Dies erfolgt in WebSphere MQ Classes for JMS, indem die ganzzahlige Eigenschaft JMS_IBM_RETAIN auf den Wert 1 gesetzt wird. 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 WebSphere MQ Classes for JMS

In den Bindungs- und Clientmodi unterstützt JMS XA-kompatible Transaktionen mit einem unterstützten Transaktionsmanager.

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.

Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker verwenden

Eine WebSphere MQ Classes for JMS-Anwendung kann eine Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker für das Publish/Subscribe-Messaging verwenden. Sowohl der Broker als auch WebSphere MQ Classes for JMS müssen so konfiguriert sein, dass eine Echtzeitverbindung möglich ist.

Wenn eine Anwendung eine Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker verwendet, wird WebSphere MQ Real-Time Transport verwendet, um Nachrichten zwischen der Anwendung und dem Broker auszutauschen. Je nach Konfiguration kann auch WebSphere MQ Multicast Transport verwendet werden, um Nachrichten an die Anwendung zu übermitteln.

Informationen dazu, wie eine Anwendung eine Verbindung zu einem WebSphere MQ-Warteschlangenmanager herstellen und WebSphere MQ Enterprise Transport verwenden kann, um Nachrichten mit einem Broker von WebSphere Event Broker oder WebSphere Message Broker auszutauschen, finden Sie in der Dokumentation für frühere Releases von WebSphere MQ Classes for JMS. Beachten Sie, dass eine Anwendung, wenn WebSphere MQ Enterprise Transport verwendet werden soll, die Verbindung zu einem Warteschlangenmanager mit einer Verbindungsfactory herstellen muss, die im Migrationsmodus des WebSphere MQ-Messaging-Providers ausgeführt wird.

Broker von WebSphere Event Broker oder WebSphere Message Broker für eine Echtzeitverbindung konfigurieren

Damit eine WebSphere MQ Classes for JMS-Anwendung eine Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker verwenden kann, müssen Sie den Broker konfigurieren, indem Sie einen Nachrichtenfluss zum Lesen der Nachrichten vom TCP/IP-Port, den der Broker überwacht, und Veröffentlichen der Nachrichten erstellen und implementieren. Je nach Ihren Anforderungen müssen Sie möglicherweise den Broker auf verschiedene Arten konfigurieren.

Um den Broker zu konfigurieren, müssen Sie einen der folgenden Nachrichtenflüsse erstellen und implementieren:

- Einen Nachrichtenfluss, der einen Real-timeOptimizedFlow-Nachrichtenverarbeitungsknoten enthält
- Einen Nachrichtenfluss, der einen Real-timeInput-Nachrichtenverarbeitungsknoten und einen Publication-Nachrichtenverarbeitungsknoten enthält

Den Real-timeOptimizedFlow- bzw. Real-timeInput-Knoten müssen Sie so konfigurieren, dass der TCP/IP-Port, der für Echtzeitverbindungen verwendet wird, überwacht wird. Standardmäßig wird für Echtzeitverbindungen die Portnummer 1506 verwendet.

Wenn Sie eine der folgenden Anforderungen stellen, müssen Sie auch den Broker konfigurieren:

- Wenn die Anwendung die Verbindung zum Broker mit SSL-Authentifizierung (SSL = Secure Sockets Layer) herstellen soll
- Wenn die Anwendung die Verbindung zum Broker mit HTTP-Tunnelung herstellen soll
- Wenn Nachrichten einem Nachrichtenkonsumenten mithilfe von Multicasting zugestellt werden sollen

Informationen zur Konfiguration eines Brokers finden Sie in der *WebSphere Event Broker-Produktdokumentation* oder *WebSphere Message Broker-Produktdokumentation*.

WebSphere MQ-Klassen für JMS für eine Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker konfigurieren

Damit eine Anwendung von WebSphere MQ-Klassen für JMS eine Echtzeitverbindung zu einem Broker von WebSphere Event Broker oder WebSphere Message Broker verwendet, muss WebSphere MQ-Klassen für JMS konfiguriert werden, indem bestimmte Eigenschaften der Verbindungsfactory eingestellt werden. Abhängig von Ihren Voraussetzungen muss WebSphere MQ-Klassen für JMS möglicherweise auf zusätzliche Arten konfiguriert werden.

Um WebSphere MQ-Klassen für JMS zu konfigurieren, müssen folgende Eigenschaften der Verbindungsfactory festgelegt werden:

- Die Eigenschaft TRANSPORT muss auf DIRECT gesetzt werden.

  Damit allerdings eine Anwendung eine Verbindung mittels HTTP-Tunnelling herstellt, muss die Eigenschaft TRANSPORT auf DIRECTHTTP eingestellt werden. Siehe „Verwendung von HTTP-Tunnelling” auf Seite 983.
- Die Eigenschaft HOSTNAME muss auf den Hostnamen oder die IP-Adresse des Systems eingestellt werden, auf dem der Broker ausgeführt wird.
- Die Eigenschaft PORT muss auf die Nummer des Port eingestellt werden, auf dem der Broker für Echtzeitverbindungen empfangsbereit ist.

Eine Anwendung kann diese Eigenschaften dynamisch zur Laufzeit mithilfe der IBM JMS-Erweiterungen oder der WebSphere MQ-JMS-Erweiterungen festlegen. Wenn die Verbindungsfactory ein verwaltetes Objekt ist, kann ein Administrator alternativ diese Eigenschaften mithilfe des WebSphere MQ-JMS-Verwaltungstools oder WebSphere MQ Explorer festlegen.

Informationen über Eigenschaften und die Methoden, die von Anwendungen zum Einstellen ihrer Werte verwendet werden, finden Sie unter Eigenschaften von IBM WebSphere MQ classes for JMS-Objekten. Informationen zur Verwendung des WebSphere MQ JMS-Verwaltungstools erhalten Sie unter „Einsatz des WebSphere MQ JMS-Verwaltungstools” auf Seite 992. Informationen zur Verwendung von WebSphere MQ Explorer finden Sie in der bei WebSphere MQ Explorer bereitgestellten Hilfe.

Falls eine der folgenden Anforderungen vorliegt, erfordert WebSphere MQ-Klassen für JMS eine zusätzliche Konfiguration:

- Wenn eine Anwendung mithilfe der SSL-Authentifizierung (Secure Sockets Layer) eine Verbindung zum Broker herstellen soll
- Wenn eine Anwendung mithilfe von HTTP-Tunnelling eine Verbindung zum Broker herstellen soll
- Wenn eine Anwendung über einen Proxy-Server eine Verbindung zum Broker herstellen soll
- Wenn Nachrichten einem Nachrichtenkonsumenten mithilfe von Multicasting zugestellt werden sollen

In den folgenden Abschnitten wird beschrieben, wie WebSphere MQ-Klassen für JMS für jede dieser Voraussetzungen konfiguriert wird.

Verwendung von SSL-Authentifizierung (Secure Sockets Layer)

SSL-Authentifizierung kann bei einer Echtzeitverbindung zu einem Broker verwendet werden. Ausschließlich Authentifizierung wird für diese Verbindungsart unterstützt. SSL kann nicht zum Verschlüsseln und Entschlüsseln der Nachrichtendaten, die zwischen der Anwendung und dem Broker fließen, oder zum Erkennen von Datenmanipulation verwendet werden.

Beachten Sie den Unterschied zwischen dieser Situation und derjenigen, wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt. Im letzteren Fall können Sie die WebSphere MQ SSL-Unterstützung zum Verschlüsseln und Entschlüsseln der Nachrichtendaten, die zwischen der Anwendung und dem Warteschlangenmanager fließen, zum Erkennen von Datenmanipulation sowie zur Bereitstellung von Authentifizierung verwenden.

Wenn Sie Nachrichtendaten bei einer Echtzeitverbindung zu einem Broker schützen wollen, können Sie stattdessen die vom Broker bereitgestellte Funktion verwenden. Sie können einen Datenschutzniveau-Wert (QoP) für jedes Thema mit Nachrichten zuweisen, die Sie schützen möchten. Aus diesem Grund können Sie für jedes Thema eine andere Nachrichtenschutzstufe auswählen. Weitere Informationen zum Nachrichtenschutz, der von einem Broker bereitgestellt wird, erhalten Sie in der *WebSphere Event Broker-Produktdokumentation* oder *WebSphere Message Broker-Produktdokumentation*.

Um SSL-Authentifizierung bei einer Echtzeitverbindung zu einem Broker zu verwenden, muss die Eigenschaft DIRECTAUTH der Verbindungsfactory auf CERTIFICATE festgelegt werden.

Wenn Sie SSL für gegenseitige Authentifizierung verwenden wollen, muss die Authentifizierungsprotokolltyp-Eigenschaft des Brokers die Option R für symmetrisches SSL angeben. Wenn Sie SSL nur für die Authentifizierung des Brokers verwenden wollen, muss die Authentifizierungsprotokolltyp-Eigenschaft des Brokers die Option S für asymmetrisches SSL angeben. Aber in diesem Fall muss die Anwendung eine Verbindung zum Broker herstellen, indem sie createConnection() mit einer Benutzer-ID und einem Kennwort als Parameter aufruft (s. Beispiel unten):

```
factory.createConnection("user1", "user1pw");
```

Der Broker verwendet dann die Benutzer-ID und das Kennwort anstelle von SSL, um die Anwendung zu authentifizieren. Weitere Informationen zur Konfiguration des Brokers für die SSL-Authentifizierung erhalten Sie in der *WebSphere Event Broker-Produktdokumentation* oder *WebSphere Message Broker-Produktdokumentation*.

Anmerkungen:

1. Der Wert der Eigenschaft DIRECTAUTH bestimmt, ob SSL-Authentifizierung bei einer Echtzeitverbindung zu einem Broker verwendet wird, nicht der Wert der Eigenschaft SSLCIPHERSUITE.
2. Wenn SSL-Authentifizierung bei einer Echtzeitverbindung zu einem Broker verwendet wird, werden mithilfe der Eigenschaften SSLPEERNAME und SSLCRL die gleichen Prüfungen durchgeführt wie bei einer Anwendung, die eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt.
3. WebSphere MQ Classes for JMS kann dieselbe JSSE-Keystore-und -Truststore-Konfiguration (Java Secure Socket Extension) verwenden, um die SSL-Unterstützung in einer der folgenden Situationen bereitzustellen:
   - Wenn eine Anwendung eine Echtzeitverbindung zu einem Broker verwendet
   - Wenn eine Anwendung eine Verbindung zu einem Warteschlangenmanager im Clientmodus herstellt

Verwendung von HTTP-Tunnelling

Eine Anwendung von WebSphere MQ-Klassen für JMS kann eine Verbindung zu einem Broker mittels HTTP-Tunnelling herstellen, d. h. die Anwendung verwendet das HTTP-Protokoll wie bei einer Verbindung zu einer Website, um eine Verbindung zum Broker herzustellen.

Um HTTP-Tunnelling bei einer Echtzeitverbindung zu einem Broker zu verwenden, muss die Eigenschaft TRANSPORT der Verbindungsfactory auf DIRECTHTTP eingestellt werden.

HTTP-Tunnelling kann nicht gemeinsam mit SSL-Authentifizierung bei einer Verbindung über einen Proxy-Server oder zur Zustellung von Nachrichten mittels Multicasting verwendet werden. Die unterstützte HTTP-Protokoll-Version ist 1.0. HTTP-Version 1.1 wird nicht unterstützt.

Verbindung über einen Proxy-Server

Eine Anwendung von WebSphere MQ-Klassen für JMS kann eine Echtzeitverbindung zu einem Broker über einen Proxy-Server verwenden. WebSphere MQ Classes for JMS stellt eine direkte Verbindung zum Proxy-Server her und verwendet das in RFC 2817 definierte Internetprotokoll, um den Proxy-Server aufzufordern, die Verbindungsanforderung an den Broker weiterzuleiten.

Um eine Verbindung zu einem Broker über einen Proxy-Server herzustellen, müssen die folgenden Eigenschaften der Verbindungsfactory festgelegt werden:

- Die Eigenschaft PROXYHOSTNAME muss auf den Hostnamen oder die IP-Adresse des Systems eingestellt werden, auf dem der Broker ausgeführt wird.
- Die Eigenschaft PROXYPORT muss auf die Nummer des Port eingestellt werden, auf dem der Proxy-Server empfangsbereit ist.

Wenn die Eigenschaft PROXYHOSTNAME nicht oder auf die leere Zeichenfolge festgelegt wird, versucht WebSphere MQ-Klassen für JMS eine direkte Verbindung zum Broker herzustellen, indem es nur die Eigenschaften HOSTNAME und PORT verwendet, und versucht keinen Verbindungsaufbau über einen Proxy-Server.

Zustellung von Nachrichten mittels Multicasting

Durch die Verwendung einer Echtzeitverbindung zu einem Broker können Nachrichten mittels Multicasting an einen Nachrichtenkonsumenten zugestellt werden.

Um Multicasting zu aktivieren, muss die Eigenschaft MULTICAST des Topic-Objekts auf die erforderliche Multicast-Option eingestellt werden. Alternativ hierzu muss, wenn die Eigenschaft MULTICAST des Topic-Objekts auf ASCF eingestellt wird, die Eigenschaft MULTICAST der Verbindungsfactory auf die erforderliche Multicast-Option festgelegt werden.

WebSphere MQ-Klassen für JMS unterstützt die Multicast-Protokolle PTL (Packet Transfer Layer) und PGM (Pragmatic General Multicast) und beinhaltet Unterstützung für beide Implementierungen des PGM-Protokolls, eingebundenes PGM/IP und PGM UDP. Allerdings ist PGM/IP-Unterstützung nur auf den folgenden Plattformen verfügbar:

- AIX (nur 32-Bit)
- Linux (x86-Plattform)
- Linux (zSeries -Plattform, nur 32 Bit)
- Solaris SPARC (nur 32-Bit)
- Windows (nur 32-Bit)
- z/OS

WebSphere MQ-Klassen für JMS - Anwendungsserverfunktionen

In diesem Abschnitt wird beschrieben, wie WebSphere MQ-Klassen für JMS die ConnectionConsumer-Klasse und erweiterte Funktionalität in der Session-Klasse implementiert. Darüber hinaus enthält der Abschnitt eine Zusammenfassung der Funktion eines Serversitzungspools.

WebSphere MQ Classes for JMS unterstützt Application Server Facilities (ASF), die in der *Java Message Service Specification Version 1.1* angegeben sind (siehe die Java-Website von Sun unter https://java.sun.com). Diese Spezifikation gibt drei Rollen innerhalb dieses Programmiermodells an:

- **Der JMS-Provider** stellt die ConnectionConsumer- und erweiterte Session-Funktionalität 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 Topic eintrifft, wählt JMS einen Thread aus dem Pool und stellt ihm ein Batch Nachrichten zu. 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 985
- „Allgemeine Grundsätze für das Publish/Subscribe-Messaging unter Verwendung von ASF” auf Seite 986
- „Nachrichten aus der Warteschlange in ASF entfernen” auf Seite 987
- Behandlung von nicht verarbeitbaren Nachrichten in ASF. (siehe „Handhabung nicht verarbeitbarer Nachrichten in IBM WebSphere MQ classes for JMS” auf Seite 945).

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 Connection-Consumer 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 WebSphere 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 WebSphere 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 987). 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.

Wenn Sie den WebSphere MQ-Warteschlangenmanager einrichten, berücksichtigen Sie 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. Details erhalten Sie in der WebSphere MQ-Dokumentation für Ihre spezifische 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.

Einzelheiten zu MQSC-Befehlen entnehmen Sie der WebSphere MQ-Scriptbefehlsreferenz (MQSC).

Allgemeine Grundsätze für das Publish/Subscribe-Messaging unter Verwendung von ASF

ConnectionConsumers erhalten Nachrichten zu einem angegebenen Thema (Topic). Ein ConnectionConsumer kann permanent oder nicht permanent sein. Sie müssen angeben, welche Warteschlange oder Warteschlangen das ConnectionConsumer verwendet.

Wenn eine Anwendung einen ConnectionConsumer aus einem TopicConnection-Objekt erstellt, gibt sie ein Topic-Objekt und eine Selektorzeichenfolge an. Der ConnectionConsumer beginnt dann mit dem Empfang von Nachrichten, die mit dem Selektor für das betreffende Thema übereinstimmen. Dazu gehören auch ständige Veröffentlichungen zu dem subkribierten Thema.

Alternativ dazu kann eine Anwendung einen permanenten ConnectionConsumer erstellen, der einem bestimmten Namen zugeordnet ist. Dieser ConnectionConsumer empfängt Nachrichten, die im Thema veröffentlicht wurden, seit der permanente ConnectionConsumer zuletzt aktiv war. Er empfängt alle derartigen Nachrichten, die dem Selektor für das Thema entsprechen. Wenn der ConnectionConsumer jedoch das Vorauslesen verwendet, kann er nicht persistente Nachrichten verlieren, die sich beim Schließen im Clientpuffer befinden.

Wenn sich WebSphere MQ-Klassen für JMS im Migrationsmodus des WebSphere 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 WebSphere MQ-Klassen für JMS im Migrationsmodus des WebSphere 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.

Wenn Sie den WebSphere MQ-Warteschlangenmanager einrichten, berücksichtigen Sie 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. Details erhalten Sie in der WebSphere MQ-Dokumentation für Ihre 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 trifft möglicherweise eine Nachricht ein, dass JMS nicht parsen kann.

Nicht verarbeitbare Nachricht
Möglicherweise erreicht eine Nachricht den Rücksetzschwellenwert, der ConnectionConsumer kann sie jedoch nicht neu in die Rücksetzwarteschlange einreihen.

Kein interessierter ConnectionConsumer
Wenn die QueueConnectionFactory beim Punkt-zu-Punkt-Messaging so festgelegt ist, dass sie nicht erwünschte Nachrichten nicht behält, trifft eine Nachricht ein, die von keinem der ConnectionConsumers gewünscht wird.

In diesen Situationen versucht der ConnectionConsumer, die Nachricht aus der Warteschlange zu entfernen. Die Dispositionsoptionen im Berichtsfeld des MQMD der Nachricht bestimmen das genaue Verhalten. Diese Optionen sind:

MQRO_DEAD_LETTER_Q
Die Nachricht wird neu in die Warteschlange für nicht zustellbare Nachrichten des Warteschlangenmanagers eingereiht. Dies ist die Standardeinstellung.

MQRO_DISCARD_MSG
Die Nachricht wird gelöscht.

Der ConnectionConsumer generiert auch eine Berichtsnachricht. Dieses Verhalten hängt ebenfalls vom Berichtsfeld des MQMD der Nachricht ab. Diese Nachricht wird an die Antwortwarteschlange (ReplyToQ) im ReplyToQmgr gesendet. Falls beim Senden der Berichtsnachricht ein Fehler auftritt, wird die Nachricht stattdessen an die Warteschlange für nicht zustellbare Nachrichten gesendet. Die Ausnahmeberichtsoptionen im Berichtsfeld des MQMD der Nachricht legen Details der Berichtsnachricht fest. Diese Optionen sind:

MQRO_EXCEPTION
Eine Berichtsnachricht wird generiert, die den MQMD der ursprünglichen Nachricht enthält. Sie enthält keinen Nachrichteninhalt.

MQRO_EXCEPTION_WITH_DATA
Eine Berichtsnachricht wird generiert, die den MQMD, alle MQ-Header und 100 Byte der Nachrichtendaten enthält.

MQRO_EXCEPTION_WITH_FULL_DATA
Eine Berichtsnachricht wird generiert, die alle Daten aus der ursprünglichen Nachricht enthält.

Standard
Es wird keine Berichtsnachricht generiert.

Bei der Generierung von Berichtsnachrichten werden die folgenden Optionen berücksichtigt:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Wenn eine nicht verarbeitbare Nachricht nicht neu eingereiht werden kann, da möglicherweise die Warteschlange für nicht zustellbare Nachrichten voll oder die Berechtigung falsch angegeben ist, hängt die jeweilige Aktion von der Persistenz der Nachricht ab. Wenn die Nachricht nicht persistent ist, wird die Nachricht gelöscht und es wird keine Berichtsnachricht generiert. Wenn die Nachricht persistent ist, wird die Nachrichtenübermittlung an alle Verbindungskonsumenten gestoppt, die an dem betreffenden Ziel empfangsbereit sind. Diese Verbindungskonsumenten müssen geschlossen werden. Das Problem muss gelöst werden, bevor sie wieder erstellt werden können und ein Neustart der Nachrichtenübermittlung möglich ist.

Sie müssen unbedingt eine Warteschlange für nicht zustellbare Nachrichten definieren und diese regelmäßig prüfen, um sicherzustellen, dass keine Probleme auftreten. Vor allem müssen Sie darauf achten, dass die Warteschlange für nicht zustellbare Nachrichten ihre maximale Tiefe nicht erreicht. Außerdem muss ihre maximale Nachrichtengröße für alle Nachrichten ausreichen.

Wenn eine Nachricht neu in die Warteschlange für nicht zustellbare Nachrichten eingereiht wird, wird ihr ein WebSphereMQ-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 989. 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 WebSphere MQ-Klassen für JMS ab.

Informationen zur Behandlung nicht verarbeitbarer Nachrichten in WebSphere MQ-Klassen für JMS finden Sie im Abschnitt „Handhabung nicht verarbeitbarer Nachrichten in IBM WebSphere MQ classes for JMS” auf Seite 945.

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.

Der integrierte JMS-Provider in WebSphere Application Server Version 5.0 und Version 5.1 behandelt nicht verarbeitbare Nachrichten anders als für WebSphere MQ-Klassen für JMS beschrieben. Sie finden Informationen darüber, wie der integrierte JMS-Provider nicht verarbeitbare Nachrichten behandelt, in der jeweiligen Produktdokumentation zu WebSphere Application Server.

Fehlerbehandlung

In diesem Abschnitt werden verschiedene Aspekte der Fehlerbehandlung beschrieben, darunter„Wiederherstellung nach Fehlerbedingungen in ASF" auf Seite 989 und „Ursachen- und Feedbackcodes in ASF" auf Seite 989.

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. Anhand der API-Ursachencodes können Sie die Ursache dieser Fehler diagnostizieren.

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 165 auf Seite 991 werden die Grundsätze der ServerSessionPool- und ServerSession-Funktionalität zusammengefasst.

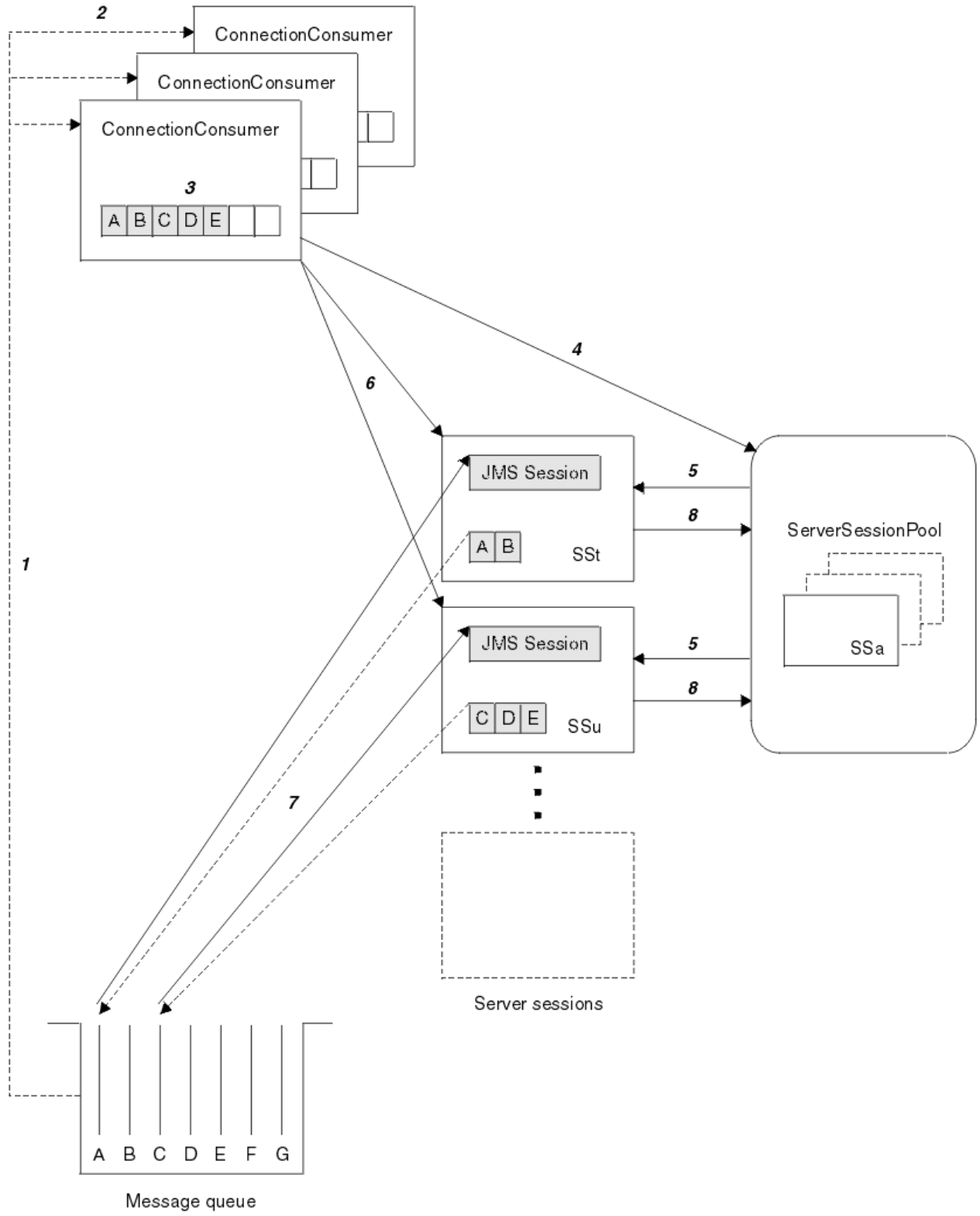


Abbildung 165. ServerSessionPool- und ServerSession-Funktionalität

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.

Einsatz des WebSphere MQ JMS-Verwaltungstools

Das Verwaltungstool dient dazu, die Eigenschaften der acht Typen von WebSphere MQ-Klassen für ein JMS-Objekt zu definieren und in einem JNDI-Namensbereich zu speichern. Anwendungen können dann JNDI verwenden, um diese verwalteten Objekte aus dem Namespace abzurufen.

Die WebSphere MQ-Klassen für JMS-Objekte, die Sie mithilfe des Tools verwalten können, sind:

- MQConnectionFactory
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

Details zu diesen Objekten erhalten Sie unter „JMS-Objekte verwalten” auf Seite 997.

Die Eigenschaftstypen und Werte, die Sie für die Verwendung dieses Tools benötigen, werden unter Eigenschaften von IBM WebSphere MQ classes for JMS-Objekten aufgeführt.

Darüber hinaus gibt das Tool Administratoren die Möglichkeit, Subkontexte eines Verzeichnisnamensbereichs innerhalb von JNDI zu bearbeiten. Weitere Informationen hierzu erhalten Sie unter „Subkontexte mit dem WebSphere MQ JMS-Verwaltungstool bearbeiten” auf Seite 996.

Sie können auch JMS-verwaltete Objekte mit dem WebSphere MQ Explorer erstellen und konfigurieren.

Verwaltungstool der IBM WebSphere MQ classes for JMS aufrufen

Das Verwaltungstool verfügt über eine Befehlszeilenschnittstelle. Sie können diese interaktiv verwenden oder damit einen Stapelprozess starten.

Der interaktive Modus stellt eine Eingabeaufforderung zur Verfügung, in der Sie Verwaltungsbefehle eingeben können. Im Stapelbetrieb enthält der Befehl zum Starten des Tools den Namen einer Datei, die ein Verwaltungsbefehlsscript enthält.

Interaktiver Modus

Um das Tool im interaktiven Modus zu starten, geben Sie diesen Befehl ein:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

Dabei gilt:

-t
Aktiviert den Trace (der Standardwert ist 'trace off').

Die Tracedatei wird in "%MQ_JAVA_DATA_PATH%"\errors (Windows) bzw. /var/mqm/trace (UNIX) generiert. Der Name der Tracedatei hat das folgende Format:

`mqjms_PID.trc`

Dabei steht *PID* für die Prozess-ID der JVM.

-v
Generiert eine ausführliche Ausgabe (Standardeinstellung ist 'terse output')

-cfg Konfigurationsdateiname
Benennt eine alternative Konfigurationsdatei. Wenn dieser Parameter ausgeschlossen wird, wird die Standardkonfigurationsdatei, JMSAdmin.config, verwendet. (Weitere Informationen erhalten Sie unter „JMS-Verwaltungstool konfigurieren" auf Seite 993.)

Es wird eine Eingabeaufforderung angezeigt, die angibt, dass das Tool bereit ist, Verwaltungsbefehle zu akzeptieren. Diese Eingabeaufforderung wird zunächst wie folgt angezeigt:

```
InitCtx>
```

Dies gibt an, dass der aktuelle Kontext (d. h. der JNDI-Kontext, auf den sich alle Benennungs- und Verzeichnisoperationen aktuell beziehen) der ursprüngliche Kontext ist, der im Konfigurationsparameter PROVIDER_URL definiert ist (weitere Informationen erhalten Sie unter „JMS-Verwaltungstool konfigurieren" auf Seite 993).

Wenn Sie den Verzeichnisnamensbereich durchlaufen, ändert sich die Eingabeaufforderung, sodass die Eingabeaufforderung immer den aktuellen Kontext anzeigt.

Stapelbetrieb

Um das Tool im Stapelmodus zu starten, geben Sie diesen Befehl ein:

```
JMSAdmin <test.scp
```

Dabei ist *test.scp* eine Scriptdatei, die Administrationsbefehle enthält (weitere Informationen hierzu erhalten Sie unter „Verwaltungsbefehle im WebSphere MQ JMS-Verwaltungstool" auf Seite 995). Der letzte Befehl in der Datei muss der Befehl END sein.

JMS-Verwaltungstool konfigurieren

Das WebSphere MQ JMS-Verwaltungstool verwendet für die Festlegung der Werte bestimmter Eigenschaften eine Konfigurationsdatei. Es wird eine Beispieldatei bereitgestellt, die Sie an Ihr System anpassen können.

Die Konfigurationsdatei ist eine Textdatei, die aus einer Gruppe von Schlüssel/Wert-Paaren besteht, die durch das Gleichheitszeichen (=) voneinander getrennt sind. Dies wird im folgenden Beispiel gezeigt:

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(A # in der ersten Spalte der Zeile gibt einen Kommentar an oder eine Zeile, die nicht verwendet wird.)

Es wird eine Beispielkonfigurationsdatei bei WebSphere MQ bereitgestellt. Die Datei heißt `JMSAdmin.config`und befindet sich im Verzeichnis `<MQ_JAVA_INSTALL_PATH>/bin` . Bearbeiten Sie diese Datei passend zur Konfiguration Ihres Systems.

Konfigurieren Sie das Verwaltungstool mit Werten für die folgenden Eigenschaften:

INITIAL_CONTEXT_FACTORY
Der Service-Provider, den das Tool verwendet. Die unterstützten Werte für diese Eigenschaft lauten wie folgt:

- com.sun.jndi.ldap.LdapCtxFactory (für LDAP)

- com.sun.jndi.fscontext.RefFSContextFactory (für Dateisystemkontext)

Sie können auch eine Ausgangskontextfactory verwenden, die sich nicht in der vorangehenden Liste befindet. Weitere Informationen finden Sie in „Einsatz einer nicht aufgeführten Ausgangskontextfactory mit dem WebSphere MQ JMS-Verwaltungstool” auf Seite 994.

PROVIDER_URL
Die URL des Ausgangskontextes der Sitzung; das Stammverzeichnis aller vom Tool ausgeführten JNDI-Operationen. Es werden zwei Formen dieser Eigenschaft unterstützt:

- ldap://hostname/kontextname
- file: [Laufwerk:] /Pfadname

Das Format der LDAP-URL kann abhängig von Ihrem LDAP-Provider variieren. Weitere Informationen finden Sie in der LDAP-Dokumentation.

SECURITY_AUTHENTICATION
Ob JNDI Ihrem Serviceanbieter Sicherheitsberechtigungsnachweise übergibt. Diese Eigenschaft wird nur verwendet, wenn ein LDAP-Service-Provider genutzt wird. Diese Eigenschaft kann eine von drei Werten aufweisen:

- none (anonyme Authentifizierung)
- simple (einfache Authentifizierung)
- CRAM-MD5 (CRAM-MD5-Authentifizierungsmechanismus)

Wenn ein gültiger Wert nicht bereitgestellt wird, nimmt die Eigenschaft standardmäßig den Wert 'none' an. Weitere Informationen zur Sicherheit beim Verwaltungstool erhalten Sie unter „Sicherheit für das JMS-Verwaltungstool konfigurieren” auf Seite 995.

Diese Eigenschaften werden in einer Konfigurationsdatei festgelegt. Wenn Sie das Tool aufrufen, können Sie diese Konfiguration mithilfe des Befehlszeilenparameters `-cfg` angeben, wie in „Verwaltungstool der IBM WebSphere MQ classes for JMS aufrufen” auf Seite 992 beschrieben. Wenn Sie keinen Konfigurationsdateinamen angeben, versucht das Tool, die Standardkonfigurationsdatei (`JMSAdmin.config`) zu laden. Sie sucht zuerst im aktuellen Verzeichnis nach dieser Datei und dann im Verzeichnis `<MQ_JAVA_INSTALL_PATH>/bin` , wobei `<MQ_JAVA_INSTALL_PATH>` der Pfad zu Ihrer WebSphere MQ Classes for JMS-Installation ist.

Einsatz einer nicht aufgeführten Ausgangskontextfactory mit dem WebSphere MQ JMS-Verwaltungstool

Es werden zwei Ausgangskontextfactory-Werte unterstützt. Sie können weitere JNDI-Kontexte verwenden, indem Sie die Parameter in der Konfigurationsdatei der JMS-Verwaltung festlegen.

Sie können das Verwaltungstool für Verbindungen zu JNDI-Kontexten verwenden, die nicht in „JMS-Verwaltungstool konfigurieren” auf Seite 993 aufgeführt werden. Verwenden Sie hierfür drei Parameter, die in der JMSAdmin-Konfigurationsdatei definiert sind.

Gehen Sie wie folgt vor, um eine andere InitialContextFactory zu verwenden

1. Legen Sie für die Eigenschaft INITIAL_CONTEXT_FACTORY den erforderlichen Klassennamen fest.
2. Definieren Sie das Verhalten der Ausgangskontextfactory mithilfe der Eigenschaften USE_INITIAL_DIR_CONTEXT, NAME_PREFIX und NAME_READABILITY_MARKER.

Die Einstellungen für diese Eigenschaften werden in den Kommentaren der Beispielkonfigurationsdatei beschrieben.

Sie müssen die drei hier aufgeführten Eigenschaften nicht definieren, wenn Sie einen der unterstützten INITIAL_CONTEXT_FACTORY-Werte verwenden. Sie können ihnen jedoch Werte geben, um die Systemstandardwerte außer Kraft zu setzen. Wenn Sie eine oder mehrere der drei Eigenschaften von InitialContextFactory weglassen, stellt das Verwaltungstool geeignete Standardwerte bereit, die auf den Werten der anderen Eigenschaften basieren.

Sicherheit für das JMS-Verwaltungstool konfigurieren

Bestimmen Sie mit der Eigenschaft SECURITY_AUTHENTICATION, ob Sicherheitsberechtigungsnachweise an den Service-Provider übergeben werden.

Die Eigenschaft SECURITY_AUTHENTICATION wird unter „JMS-Verwaltungstool konfigurieren" auf Seite 993 beschrieben. Sie wirkt sich wie folgt aus:

- Wenn Sie diesen Parameter auf `none` setzen, gibt JNDI keine Sicherheitsberechtigungsnachweise an den Service-Provider aus, und die *anonyme Authentifizierung* wird ausgeführt.
- Wenn Sie für den Parameter entweder `simple` oder `CRAM-MD5` festlegen, werden Sicherheitsberechtigungsnachweise über JNDI an den zugrundeliegenden Service-Provider übergeben. Diese Sicherheitsberechtigungsnachweise sind in Form eines definierten Benutzernamens (Benutzer-DN) und eines Kennworts vorhanden.

Wenn Sicherheitsberechtigungsnachweise erforderlich sind, werden Sie beim Initialisieren des Tools zur Eingabe aufgefordert. Dies können Sie vermeiden, indem Sie die Eigenschaften PROVIDER_USERDN und PROVIDER_PASSWORD in der JMSAdmin-Konfigurationsdatei festlegen.

Anmerkung: Wenn Sie diese Eigenschaften nicht verwenden, wird der eingegebene Text *einschließlich des Kennworts* an die Anzeige zurückgemeldet. Dies kann Sicherheitsauswirkungen haben.

Das Tool selbst führt keine Authentifizierung durch; Die Aufgabe wird an den LDAP-Server delegiert. Der LDAP-Serveradministrator muss Zugriffsberechtigungen für verschiedene Teile des Verzeichnisses einrichten und verwalten. Weitere Informationen finden Sie in der LDAP-Dokumentation. Wenn die Authentifizierung fehlschlägt, zeigt das Tool eine entsprechende Fehlernachricht an und wird beendet.

Ausführlichere Informationen zu Sicherheit und JNDI finden Sie in der Dokumentation auf der Sun-Java-Website (https://java.sun.com).

Verwaltungsbefehle im WebSphere MQ JMS-Verwaltungstool

Das Verwaltungstool akzeptiert Befehle, die aus einem Verwaltungsverb und seinen entsprechenden Parametern bestehen.

Wenn die Eingabeaufforderung angezeigt wird, ist das Tool bereit, Befehle zu akzeptieren. Verwaltungsbefehle liegen normalerweise im folgenden Format vor:

```
verb [param]*
```

Dabei steht **verb** für eines der in Tabelle 133 auf Seite 995aufgelisteten Verwaltungsverben. Alle gültigen Befehle enthalten ein Verb, das am Anfang des Befehls in der Standardform oder in der Kurzform angezeigt wird.

Die Parameter, die ein Verb annehmen kann, hängen vom Verb ab. Beispiel: Das Verb `END` kann keine Parameter annehmen, aber das Verb `DEFINE` eine beliebige Anzahl. Details zu den Verben, die mindestens einen Parameter annehmen können, werden in den zugehörigen Abschnitten besprochen.

Tabelle 133. Verwaltungsverben

Verb	Kurzform	Beschreibung
ALTER	Alt	Ändern Sie mindestens eine der Eigenschaften eines verwalteten Objekts.
DEFINIER	DEF	Ein verwalttes Objekt erstellen und speichern oder einen Subkontext erstellen
ANZEIGEN	DIS	Zeigt die Eigenschaften eines oder mehrerer gespeicherter verwalteter Objekte oder den Inhalt des aktuellen Kontexts an.
DELETE	ENTF	Entfernt ein oder mehrere verwaltete Objekte aus dem Namensbereich oder entfernt einen leeren Subkontext.

Tabelle 133. Verwaltungsverben (Forts.)		
Verb	**Kurzform**	**Beschreibung**
ÄNDERUNG	CHG	Ändern Sie den aktuellen Kontext, sodass der Benutzer den Verzeichnisnamensbereich an einer beliebigen Stelle unterhalb des Ausgangskontextes durchqueren kann (anstehende Sicherheitsfreigabe)
KOPIEREN	KP	Erstellen Sie eine Kopie eines gespeicherten verwalteten Objekts und speichern Sie es unter einem alternativen Namen.
MOVE	MV	Ändern des Namens, unter dem ein verwalttes Objekt gespeichert wird
END		Verwaltungstool schließen

Bei Verb-Namen wird die Groß-/Kleinschreibung nicht beachtet.

Üblicherweise wird zum Beenden von Befehlen die Eingabetaste gedrückt. Allerdings können Sie dies umgehen, indem Sie das Pluszeichen (+) direkt vor dem Wagenrücklauf eingeben. Auf diese Weise können Sie mehrzeilweise Befehle eingeben, wie im folgenden Beispiel gezeigt:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Zeilen, die mit einem der folgenden Zeichen beginnen, werden als Kommentare behandelt und ignoriert: * # / .

Subkontexte mit dem WebSphere MQ JMS-Verwaltungstool bearbeiten

Verwenden Sie die Verben **CHANGE**, **DEFINE**, **DISPLAY** und **DELETE**, um die Subkontexte des Verzeichnisnamensbereichs zu bearbeiten.

Die Verwendung dieser Verben wird in Tabelle 134 auf Seite 996 beschrieben.

Tabelle 134. Syntax und Beschreibung der Befehle, die zum Bearbeiten von Subkontexten verwendet werden	
Befehlssyntax	**Beschreibung**
DEFINE CTX (ctxName)	Es wird versucht, einen untergeordneten Subkontext des aktuellen Kontexts zu erstellen, der den Namen ctxName hat. Gibt an, ob ein Sicherheitsverstoß vorliegt, wenn der Subkontext bereits vorhanden ist oder wenn der angegebene Name nicht gültig ist.
ANZEIGEN CTX	Zeigt den Inhalt des aktuellen Kontexts an. Administrierte Objekte werden mit a, Subkontexten mit `[D]` annotiert. Der Java-Typ jedes Objekts wird ebenfalls angezeigt.
DELETE CTX (ctxName)	Es wird versucht, den untergeordneten Kontext des aktuellen Kontextes mit dem Namen 'ctxName' zu löschen. Falls der Kontext nicht gefunden wird, ist er nicht leer, oder wenn ein Sicherheitsverstoß vorliegt.

Tabelle 134. Syntax und Beschreibung der Befehle, die zum Bearbeiten von Subkontexten verwendet werden (Forts.)	
Befehlssyntax	**Beschreibung**
CTX ÄNDERN (ctxName)	Ändert den aktuellen Kontext, so dass er sich jetzt auf den untergeordneten Kontext mit dem Namen ctxName bezieht. Es kann einer von zwei speziellen Werten von ctxName angegeben werden: **= AKTIV** wird in den übergeordneten Kontext des aktuellen Kontexts verschoben **= INIT** wird direkt in den Ausgangskontext verschoben Es ist fehlgeschlagen, wenn der angegebene Kontext nicht vorhanden ist oder wenn ein Sicherheitsverstoß vorliegt.

JMS-Objekte verwalten

Dieser Abschnitt beschreibt die acht Objekttypen, die das Verwaltungstool handhaben kann. Er enthält Details zu jeder ihrer konfigurierbaren Eigenschaften und die Verben, die zu ihrer Bearbeitung verwendet werden können.

Sie können auch JMS-verwaltete Objekte mit dem WebSphere MQ Explorer erstellen und konfigurieren.

JMS-Objekttypen

In der Tabelle sind acht Typen von verwalteten Objekten aufgeführt.

Die Schlüsselwortspalte enthält die Zeichenfolgen, die Sie für *TYPE* in den in Tabelle 136 auf Seite 998gezeigten Befehlen ersetzen können.

Tabelle 135. Die JMS-Objekttypen, die vom Verwaltungstool verarbeitet werden		
Objekttyp	**Schlüsselwort**	**Beschreibung**
MQConnectionFactory	CF	Die WebSphere MQ-Implementierung der JMS-ConnectionFactory-Schnittstelle. Dies stellt ein Factory-Objekt zum Erstellen von Verbindungen in den Punkt-zu-Punkt-und Publish/Subscribe-Domänen dar.
MQQueueConnectionFactory	QCF	Die WebSphere MQ-Implementierung der JMS-QueueConnectionFactory-Schnittstelle. Dies stellt ein Factory-Objekt zum Erstellen von Verbindungen in der Punkt-zu-Punkt-Domäne dar.
MQTopicConnectionFactory	TCF	Die WebSphere MQ-Implementierung der JMS-TopicConnectionFactory-Schnittstelle. Dies stellt ein Factory-Objekt zum Erstellen von Verbindungen in der Publish/Subscribe-Domäne dar.
MQQueue	Q	Die WebSphere MQ-Implementierung der JMS-Queue-Schnittstelle. Dies ist ein Ziel für Nachrichten in der Punkt-zu-Punkt-Domäne.

Tabelle 135. Die JMS-Objekttypen, die vom Verwaltungstool verarbeitet werden (Forts.)		
Objekttyp	**Schlüsselwort**	**Beschreibung**
MQTopic	T	Die WebSphere MQ-Implementierung der JMS-Topic-Schnittstelle. Dies stellt ein Ziel für Nachrichten in der Publish/Subscribe-Domäne dar.
MQXAConnectionFactory„1" auf Seite 998	XACF	Die WebSphere MQ-Implementierung der JMS-XAConnectionFactory-Schnittstelle. Diese stellt ein Factory-Objekt zum Erstellen von Verbindungen in der Punkt-zu-Punkt- und der Publish/Subscribe-Domäne und der Angabe, wo die Verbindungen die XA-Versionen der JMS-Klassen verwenden, dar.
MQXAQueueConnectionFactory„1" auf Seite 998	XAQCF	Die WebSphere MQ-Implementierung der JMS-XAQueueConnectionFactory-Schnittstelle. Diese stellt ein Factory-Objekt zum Erstellen von Verbindungen in der Punkt-zu-Punkt-Domäne, die die XA-Versionen der JMS-Klassen verwendet, dar.
MQXATopicConnectionFactory„1" auf Seite 998	XATCF	Die WebSphere MQ-Implementierung der JMS-XATopicConnectionFactory-Schnittstelle. Diese stellt ein Factory-Objekt zum Erstellen von Verbindungen in der Publish/Subscribe-Domäne, die die XA-Versionen der JMS-Klassen verwendet, dar.

Anmerkung:

1. Diese Klassen werden für die Verwendung durch Anbieter von Anwendungsservern bereitgestellt. Es ist unwahrscheinlich, dass sie für Anwendungsprogrammierer direkt von Nutzen sein können.

Mit JMS-Objekten verwendete Verben

Sie können die Verben `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY` und `MOVE` verwenden, um verwaltete Objekte im Verzeichnisnamensbereich zu bearbeiten.

Tabelle 136 auf Seite 998 fasst die Verwendung dieser Verben zusammen. Ersetzen Sie *TYPE* mit dem Schlüsselwort, das das erforderliche verwaltete Objekt darstellt, wie in Tabelle 135 auf Seite 997 aufgeführt.

Tabelle 136. Syntax und Beschreibung der Befehle, die zum Bearbeiten von verwalteten Objekten verwendet werden	
Befehlssyntax	**Beschreibung**
ALTER *TYPE*(name) [Eigenschaft]*	Es wird versucht, die Eigenschaften des verwalteten Objekts mit den angegebenen Objekten zu aktualisieren. Wenn es einen Sicherheitsverstoß gibt, wenn das angegebene Objekt nicht gefunden wird oder wenn die neuen Eigenschaften nicht gültig sind, schlägt fehl.

Tabelle 136. Syntax und Beschreibung der Befehle, die zum Bearbeiten von verwalteten Objekten verwendet werden (Forts.)

Befehlssyntax	**Beschreibung**
DEFINE *TYPE*(name) [Eigenschaft]*	Versucht, ein verwaltetes Objekt des Typs *TYPE* mit den angegebenen Eigenschaften zu erstellen und unter dem Namen name im aktuellen Kontext zu speichern. Wenn der angegebene Name nicht gültig ist oder ein Objekt mit diesem Namen vorhanden ist oder wenn die angegebenen Eigenschaften nicht gültig sind, schlägt die Sicherheitsverletzung fehl.
DISPLAY *TYPE*(name)	Zeigt die Eigenschaften des verwalteten Objekts des Typs *TYPE* an, das im aktuellen Kontext unter dem Namen name gebunden ist. Gibt an, ob das Objekt nicht vorhanden ist, oder wenn ein Sicherheitsverstoß vorliegt.
DELETE *TYPE*(name)	Versucht, das verwaltete Objekt des Typs *TYPE* mit dem Namen name aus dem aktuellen Kontext zu entfernen. Gibt an, ob das Objekt nicht vorhanden ist, oder wenn ein Sicherheitsverstoß vorliegt.
COPY *TYPE*(nameA) *TYPE*(nameB)	Erstellt eine Kopie des verwalteten Objekts des Typs *TYPE* mit dem Namen nameA und nennt die Kopie nameB. Dies geschieht im Rahmen des aktuellen Kontexts. Fehlerhafte Objekte, wenn das zu kopierende Objekt nicht vorhanden ist, wenn ein Objekt mit dem Namen nameB vorhanden ist oder wenn ein Sicherheitsverstoß vorliegt.
MOVE *TYPE*(nameA) *TYPE*(nameB)	Verschiebt das verwaltete Objekt des Typs *TYPE* (benennt es um) mit dem Namen nameA zu nameB. Dies geschieht im Rahmen des aktuellen Kontexts. Gibt an, ob das Objekt, das versetzt werden soll, nicht vorhanden ist, wenn ein Objekt mit dem Namen nameB vorhanden ist oder wenn ein Sicherheitsverstoß vorliegt.

Objekte mit dem WebSphere MQ JMS-Verwaltungstool erstellen

Mit dem Befehl DEFINE können Sie Objekte erstellen und diese in einem JNDI-Namensbereich speichern.

Verwenden Sie die folgende Befehlssyntax:

```
DEFINE TYPE(name) [property]*
```

Dies ist das Verb `DEFINE` , gefolgt von einer *TYPE*`(name)` -Referenz auf verwaltete Objekte, gefolgt von null oder mehr *Eigenschaften* (siehe Eigenschaften von IBM WebSphere MQ classes for JMS -Objekten).

LDAP-Benennungsaspekte für JMS-Objekte

Um Ihre Objekte in einer LDAP-Umgebung zu speichern, müssen Sie diese Namen angeben, die bestimmten Konventionen entsprechen. Das Verwaltungstool kann Sie durch das Hinzufügen eines Standardpräfix bei der Einhaltung der Namenskonventionen unterstützen.

Eine Namenskonvention ist, dass Objekt- und Subkontextnamen ein Präfix einschließen müssen, wie `cn=` (allgemeiner Name) oder `ou=` (Organisationseinheit).

Das Verwaltungstool vereinfacht die Verwendung von LDAP-Service-Providern, indem es Ihnen erlaubt, auf Objekt-und Kontextnamen ohne Präfix zu verweisen. Wenn Sie kein Präfix angeben, fügt das Tool dem Namen, den Sie angeben, automatisch ein Standardpräfix hinzu. Für LDAP ist dies `cn=`.

Sie können das Standardpräfix ändern, indem Sie die Eigenschaft NAME_PREFIX in der JMSAdmin-Konfigurationsdatei, wie in „Einsatz einer nicht aufgeführten Ausgangskontextfactory mit dem WebSphere MQ JMS-Verwaltungstool” auf Seite 994 beschrieben, festlegen.

Dies ist im folgenden Beispiel dargestellt.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

Obwohl der angegebene Objektname (`testQueue`) kein Präfix aufweist, fügt das Tool automatisch eines hinzu, um die Einhaltung der LDAP-Namenskonvention sicherzustellen. Dementsprechend wird auch beim Befehl `DISPLAY Q(testQueue)` dieses Präfix hinzugefügt.

Möglicherweise müssen Sie Ihren LDAP-Server für das Speichern von Java-Objekten konfigurieren. Unterstützende Informationen für diese Konfiguration finden Sie in der Dokumentation zu Ihrem LDAP-Server.

Beispiele für Fehlerbedingungen beim Erstellen eines JMS-Objekts

Wenn Sie ein Objekt erstellen, kann es zu einer Reihe allgemeiner Fehlerbedingungen kommen.

Im Folgenden finden Sie Beispiele für diese Fehlerbedingungen:

CipherSpec zugeordnet zu CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

Ungültige Eigenschaft für Objekt

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

Ungültiger Typ für Eigenschaftswert

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

Eigenschaftsüberschneidungen-Client/Bindungen

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

Eigenschaftsüberschneidung-Initialisierung verlassen

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

Eigenschaftswert außerhalb des gültigen Bereichs

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

Unbekannte Eigenschaft

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

Nachstehend finden Sie Beispiele zu Fehlerbedingungen, die unter Windows auftreten können, wenn JNDI-verwaltete Objekte von einer JMS-Anwendung durchsucht werden.

1. Wenn Sie den WebSphere JNDI-Provider 'com.ibm.websphere.naming.WsnInitialContextFactory' verwenden, müssen Sie einen normalen Schrägstrich (/) verwenden, um auf verwaltete Objekte zuzugreifen, die in Subkontexten definiert sind (z. B. 'jms/MyQueueName'). Wenn Sie einen Backslash (\) verwenden, wird eine Ausnahme vom Typ "InvalidNameException" ausgelöst.
2. Wenn Sie den Sun JNDI-Provider 'com.sun.jndi.fscontext.RefFSContextFactory' verwenden, müssen Sie einen umgekehrten Schrägstrich (\) verwenden, um auf verwaltete Objekte zuzugreifen, die in Subkontexten definiert sind (z. B. 'ctx1\\fred'). Wenn Sie einen Schrägstrich (/) verwenden, wird die Ausnahmebedingung NameNotFoundException ausgelöst.

WebSphere MQ Explorer für JMS-Konfiguration verwenden

Mit der grafischen IBM WebSphere MQ Explorer-Benutzerschnittstelle können Sie JMS-Objekte auf Basis von WebSphere MQ-Objekten und WebSphere MQ-Objekte auf Basis von JMS-Objekten erstellen sowie andere WebSphere MQ-Objekte verwalten und überwachen.

Vorbereitende Schritte

Bevor Sie verwaltete JMS-Objekte mit dem WebSphere MQ Explorer erstellen und konfigurieren, fügen Sie einen Ausgangskontext hinzu, um den Stamm des JNDI-Namensbereiches zu definieren, in dem die JMS-Objekte im Benennungs- und Verzeichnisdienst gespeichert werden. Weitere Informationen finden Sie in der IBM WebSphere MQ Explorer-Benutzerhilfe für verwaltete JMS-Objekte.

Informationen zu diesem Vorgang

Sie können mit IBM WebSphere MQ Explorer folgende Aufgaben ausführen - entweder kontextabhängig auf Basis eines bereits in IBM WebSphere MQ Explorer vorhandenen Objekts oder mithilfe des Assistenten für die Erstellung neuer Objekte. In der Hilfe von WebSphere MQ Explorer erhalten Sie Beispiele der WebSphere MQ Explorer-Benutzerhilfe für einige typische Aufgaben.

Prozedur

- Erstellen Sie eine JMS-Verbindungsfactory von einem der folgenden WebSphere MQ-Objekte:
  a) Ein WebSphere MQ-Warteschlangenmanager auf Ihrem lokalen Computer oder auf einem fernen System
  b) Ein WebSphere MQ-Kanal
  c) Ein WebSphere MQ-Listener
- Fügen Sie einen WebSphere MQ-Warteschlangenmanager zu WebSphere MQ Explorer mit einer JMS-Verbindungsfactory hinzu.
- Erstellen Sie eine JMS-Warteschlange von einer WebSphere MQ-Warteschlange.
- Erstellen Sie eine WebSphere MQ-Warteschlange von einer JMS-Warteschlange.
- Erstellen Sie ein JMS-Topic von einem WebSphere MQ-Topic, bei dem es sich um ein WebSphere MQ-Objekt oder ein dynamisches Topic handeln kann.
- Erstellen Sie ein WebSphere MQ-Topic von einem JMS-Topic.

Paket WebSphere MQ Headers verwenden

Das Paket WebSphere MQ Headers stellt eine Reihe von Helper-Schnittstellen und -Klassen bereit, mit denen Sie die WebSphere MQ-Header einer Nachricht bearbeiten können. Das Paket WebSphere MQ Headers wird üblicherweise für die Ausführung von Verwaltungsservices über den Befehlsserver (mithilfe von PCF-Nachrichten (Programmable Command Format)) eingesetzt.

Informationen zu diesem Vorgang

Das Paket WebSphere MQ Headers ist in den Paketen `com.ibm.mq.headers` und `com.ibm.mq.pcf` enthalten. Sie können diese Funktion für beide der beiden alternativen APIs verwenden, die WebSphere MQ für die Verwendung in Java-Anwendungen bereitstellt:

- WebSphere MQ Classes for Java (wird auch als WebSphere MQ Headers Base Java bezeichnet).
- WebSphere MQ Classes for Java Message Service (WebSphere MQ Classes for JMS, auch WebSphere MQ JMS genannt).

WebSphere MQ Java-Basisanwendungen bearbeiten normalerweise MQMessage-Objekte und die Headerunterstützungsklassen können direkt mit diesen Objekten interagieren, da sie die Java-Basisschnittstellen von WebSphere MQ nativ verstehen.

Bei WebSphere MQ JMS handelt es sich bei den Nutzdaten einer Nachricht in der Regel um eine Zeichenfolge oder ein Byte-Array-Objekt, das mithilfe von DataInput- und DataOutput-Datenströmen bearbeitet werden kann. Mit dem Paket WebSphere MQ Headers ist eine Interaktion mit diesen Datenströmen möglich, daher können mit diesem Paket alle MQ-Nachrichten bearbeitet werden, die von WebSphere MQ JMS-Anwendungen gesendet und empfangen werden.

Obwohl das Paket WebSphere MQ Headers Referenzen auf das Java-Basispaket WebSphere MQ enthält, ist es auch für die Verwendung in WebSphere MQ JMS-Anwendungen vorgesehen und für die Verwendung in Umgebungen mit Java Platform, Enterprise Edition (Java EE) geeignet.

Eine typische Verwendungsweise des Pakets WebSphere MQ Headers ist die Bearbeitung von Verwaltungsnachrichten im PCF-Format (Programmable Command Format) in den folgenden Fällen:

- Um auf die Details einer WebSphere MQ-Ressource zuzugreifen.
- Um die Größe einer Warteschlange zu überwachen.
- Um den Zugriff auf eine Warteschlange zu blockieren.

Wenn Sie PCF-Nachrichten mit der JMS-API von WebSphere MQ verwenden, können Sie diese Art der Verwaltung von anwendungsorientierten Ressourcen innerhalb von Java EE -Anwendungen ausführen, ohne auf die Verwendung der Java-Basis-API von WebSphere MQ zurückgreifen zu müssen.

Prozedur

- Informationen zur Verwendung des Pakets WebSphere MQ Headers zum Bearbeiten von Nachrichtenheadern für WebSphere MQ -Klassen für Java finden Sie unter „Mit WebSphere MQ -Klassen für Java verwenden” auf Seite 1002.
- Hinweise zur Verwendung des Pakets WebSphere MQ Headers zur Bearbeitung von Nachrichtenheadern für JMS finden Sie unter „Verwendung mit den WebSphere MQ-Klassen für JMS” auf Seite 1003.

Mit WebSphere MQ -Klassen für Java verwenden

WebSphere MQ -Klassen für Java-Anwendungen bearbeiten normalerweise MQMessage-Objekte und die Headerunterstützungsklassen können direkt mit diesen Objekten interagieren, da sie die WebSphere MQ -Klassen für Java-Schnittstellen nativ verstehen.

Informationen zu diesem Vorgang

WebSphere MQ stellt einige Beispielanwendungen bereit, die die Verwendung des Pakets WebSphere MQ Headers mit der Java-API WebSphere MQ Base (WebSphere MQ Classes for Java) veranschaulichen.

Die Beispiele zeigen zwei Dinge:

- Wie eine PCF-Nachricht erstellt wird, um eine Verwaltungsaktion auszuführen und die Antwortnachricht auszuwerten.
- Vorgehensweise zum Senden dieser PCF-Nachricht unter Verwendung der MQ -Klassen für Java WebSphere .

Je nach der von Ihnen verwendeten Plattform werden diese Beispiele unterhalb des Verzeichnisses `pcf` im Verzeichnis `samples` oder `tools` der WebSphere MQ-Installation installiert (siehe „Installationsverzeichnisse für WebSphere MQ -Klassen für Java” auf Seite 697).

Vorgehensweise

1. Erstellen Sie eine PCF-Nachricht, um eine Verwaltungsaktion auszuführen und die Antwortnachricht auszuwerten.
2. Senden Sie diese PCF-Nachricht mit den WebSphere MQ -Klassen für Java.

Zugehörige Konzepte

„Behandlung von WebSphere MQ -Nachrichtenheadern mit WebSphere MQ -Klassen für Java” auf Seite 719
Es werden Java-Klassen bereitgestellt, die verschiedene Typen von Nachrichtenheadern darstellen. Darüber hinaus stehen zwei Helper-Klassen zur Verfügung.

„Handhabung von PCF-Nachrichten mit WebSphere MQ -Klassen für Java” auf Seite 725
Java-Klassen werden bereitgestellt, um PCF-strukturierte Nachrichten zu erstellen und zu analysieren und um das Senden von PCF-Anforderungen und das Sammeln von PCF-Antworten zu vereinfachen.

Verwendung mit den WebSphere MQ-Klassen für JMS

Wenn Sie die WebSphere MQ -Header mit den WebSphere MQ -Klassen für JMS verwenden möchten, führen Sie dieselben wichtigen Schritte wie für WebSphere MQ -Klassen für Java aus. Die PCF-Nachricht kann mit dem Paket WebSphere MQ Headers und demselben Beispielcode wie für WebSphere MQ Classes for Java auf genau dieselbe Weise erstellt und syntaktisch analysiert werden.

Informationen zu diesem Vorgang

Damit eine PCF-Nachricht über die WebSphere MQ-API gesendet werden kann, müssen die Nutzdaten der Nachricht in eine JMS-Nachricht des Typs JMSBytesMessage geschrieben und mithilfe der JMS-Standard-APIs gesendet werden. Die einzige Bedingung ist, dass die Nachricht im MQMD keinen JMS-RFH2-Header oder einen anderen Header mit bestimmten Werten enthält.

Führen Sie die folgenden Schritte aus, um eine PCF-Nachricht zu senden. Die Art und Weise, wie die PCF-Nachricht erstellt und Informationen aus der Antwortnachricht extrahiert werden, entspricht der Vorgehensweise für WebSphere MQ -Klassen für Java (siehe „Mit WebSphere MQ -Klassen für Java verwenden” auf Seite 1002).

Vorgehensweise

1. Erstellen Sie ein JMS-Warteschlangenziel, das die Warteschlange SYSTEM.ADMIN.COMMAND.QUEUE darstellt.

   WebSphere MQ JMS-Anwendungen senden die PCF-Nachrichten an das SYSTEM.ADMIN.COMMAND.QUEUEund benötigen Zugriff auf ein JMS-Zielobjekt, das diese Warteschlange darstellt. Für das Ziel müssen folgende Eigenschaften festgelegt sein:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Bei Verwendung von WebSphere Application Server müssen Sie diese Eigenschaften als angepasste 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 Nachricht des Typs JMSBytesMessage, die die korrekten MQMD-Werte enthält.

Es muss eine Nachricht des Typs JMSBytesMessage erstellt und die PCF-Nachricht in diese Nachricht geschrieben werden. Es muss eine Antwortwarteschlange erstellt werden, für die aber keine bestimmten Einstellungen erforderlich sind.

Der folgende Beispielcodeausschnitt veranschaulicht die Erstellung einer Nachricht des Typs JMSBytesMessage und das Schreiben eines com.ibm.mq.headers,pcf.PCFMessage-Objekts in diese Nachricht. Das PCFMessage-Objekt (pcfCmd) wurde zuvor mithilfe des Pakets WebSphere 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 JMS-Standard-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 859
JMS-Nachrichten setzen sich aus einem Header, Eigenschaften und einem Hauptteil zusammen. JMS definiert fünf Arten eines Nachrichtenhauptteils.

Web-Services in WebSphere MQ verwenden

Sie können IBM WebSphere MQ-Anwendungen für Web-Services mithilfe des IBM WebSphere MQ-Transports für SOAP oder der IBM WebSphere MQ-Bridge für HTTP entwickeln.

Der IBM WebSphere MQ-Transport für SOAP stellt einen JMS-Transport für SOAP bereit. Der IBM WebSphere MQ -Transport für SOAP ist auch in andere Umgebungen wie Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server integriert.

Weitere Informationen zu IBM WebSphere MQ Transport for SOAP finden Sie unter „WebSphere MQ-Transport für SOAP” auf Seite 1005.

Mit IBM WebSphere MQ Bridge for HTTP können Clientanwendungen Nachrichten mit IBM WebSphere MQ austauschen, ohne dass ein WebSphere MQ MQI-Client installiert werden muss. Sie können WebSphere MQ von jeder Plattform oder in jeder Sprache mit HTTP-Funktionalität aufrufen.

Weitere Informationen zur IBM WebSphere MQ -Bridge für HTTP finden Sie unter „WebSphere MQ Bridge for HTTP” auf Seite 1084.

Zugehörige Konzepte

„Konzepte für die Anwendungsentwicklung” auf Seite 8
Sie können verschiedene prozedurale oder objektorientierte Sprachen verwenden, um IBM WebSphere MQ-Anwendungen zu schreiben. Verwenden Sie die Links in diesem Abschnitt, um Informationen zu IBM WebSphere MQ -Konzepten zu erhalten, die für Anwendungsentwickler nützlich sind.

„Entscheiden, welche Programmiersprache verwendet werden soll” auf Seite 82
Verwenden Sie diese Informationen, um mehr über Programmiersprachen und Rahmendefinitionen, die IBM WebSphere MQ unterstützt, und Überlegungen im Zusammenhang mit deren Verwendung zu erfahren.

„IBM WebSphere MQ-Anwendungen entwerfen” auf Seite 94
Wenn Sie entschieden haben, wie Ihre Anwendungen die Vorteile von verfügbaren Plattformen und Umgebungen nutzen sollen, müssen Sie nun festlegen, wie die von WebSphere MQ bereitgestellten Funktionen verwendet werden sollen.

„WebSphere MQ-Beispielprogramme” auf Seite 100
In der folgenden Themensammlung finden Sie Informationen zur Verwendung von WebSphere MQ-Beispielprogrammen auf verschiedenen Plattformen.

„Anwendung zur Warteschlangensteuerung schreiben” auf Seite 206
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.

„Clientanwendungen schreiben” auf Seite 374
Informationen zum Schreiben von Clientanwendungen unter WebSphere MQ.

„Publish/Subscribe-Anwendungen schreiben” auf Seite 295
Beginnen Sie nun, Ihre eigenen WebSphere MQ-Publish/Subscribe-Anwendungen zu entwickeln.

„IBM WebSphere MQ-Anwendung erstellen” auf Seite 456
Dieser Abschnitt enthält Informationen zum Erstellen einer IBM WebSphere MQ-Anwendung auf anderen Plattformen.

„Programmfehler behandeln” auf Seite 581
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.

WebSphere MQ-Transport für SOAP

Der WebSphere MQ-Transport für SOAP stellt einen JMS-Transport für SOAP bereit. Der WebSphere MQ-Transport für SOAP ist auch in andere Umgebungen wie Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server integriert.

Einführung in den IBM WebSphere MQ-Transport für SOAP

Der IBM WebSphere MQ-Transport für SOAP stellt einen JMS-Transport für SOAP bereit. Der WebSphere MQ-SOAP-Sender und -Listener bieten ein Verfahren für den Aufruf von Web-Services.

Der WebSphere MQ-SOAP-Listener unterstützt Services, die von .NET Framework 1, .NET Framework 2 und Axis 1.4 gehostet werden. Der WebSphere MQ-SOAP-Sender unterstützt Web-Service-Clients, die in .NET Framework 1, .NET Framework 2, Axis 1.4 und Axis2 ausgeführt werden. Bei den Clients kann es sich um eine WebSphere MQ-Server- oder -Clientanwendung handeln. Der IBM WebSphere MQ-Transport

für SOAP ist auch in anderen Umgebungen, wie beispielsweise Microsoft Windows Communication Foundation, WebSphere Application Server und CICS Transaction Server, integriert.

Die Integration in Microsoft Windows Communication Foundation ist Bestandteil der IBM WebSphere MQ-Unterstützung für .NET Framework 3.

Bei dem IBM WebSphere MQ-Transport für SOAP handelt es sich um eine Reihe von Protokollen und Tools für den Transport von SOAP-Nachrichten mithilfe von JMS über IBM WebSphere MQ. Er wird auf verschiedene Arten für unterschiedliche Anwendungsumgebungen gepackt (siehe Tabelle 137 auf Seite 1006).

Tabelle 137. Anwendungsumgebungen mit IBM WebSphere MQ-Transport für SOAP

	In zusätzliche WebSphere MQ-Komponenten integriert	**In ein Framework integriert**
Bereitstellung als Bestandteil der WebSphere 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 WebSphere MQ-Transports für SOAP in ein Anwendungsframework vereinfacht die Entwicklung und Bereitstellung von Web-Services für IBM WebSphere MQ.

Mit zusätzlichen IBM WebSphere MQ-SOAP-Komponenten können Sie durch direkte Interaktion mit den WebSphere MQ-SOAP-Komponenten Services entwickeln und implementieren. Verwenden Sie die IBM WebSphere MQ-SOAP-Tools für die Konfiguration und Implementierung der Web-Services und Web-Service-Clients in IBM WebSphere 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. Die erforderlichen IBM WebSphere MQ-Warteschlangen, -Kanäle und -Warteschlangenmanager müssen Sie nach wie vor mit WebSphere MQ-Tools konfigurieren.

Sie können IBM WebSphere MQ-SOAP-Clients und -Server aus allen diesen Umgebungen beliebig kombinieren.

Leistungen

Der WebSphere MQ-Transport für SOAP bietet bereits vorhandenen IBM WebSphere MQ-Benutzern die folgenden grundlegenden Vorteile:

Verwendung Ihres IBM WebSphere 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 WebSphere MQ-Netz für die Verbindung von Web-Services verwenden können. Der IBM WebSphere 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 WebSphere MQ-Schnittstellen

Normalerweise muss für Anwendungen ein bestimmter WebSphere 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 wer-

den, 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 WebSphere 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 der WebSphere MQ-Transport für 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 WebSphere 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 WebSphere 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 WebSphere MQ-Implementierung der SOAP/JMS-Bindung verwendet einen URI, der eines der folgenden beiden Formate hat:

  WebSphere MQ-Transport für SOAP

  ```
  jms:/queue?&Name=Value&Name=Value...
  ```

  WebSphere MQ-Sendeformat für W3C-Kandidatenempfehlung

  ```
  jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
  ```

- Die Zuordnung einer SOAP-Nachricht zu einer WebSphere MQ-Nachricht.
- Zwei IBM WebSphere MQ -SOAP-Listener für den Empfang von SOAP-Anforderungen, einer für Java und einer für .NET Framework 1 oder .NET Framework 2. Die Listener verwenden .NET oder Axis 1.4 für die Verarbeitung der SOAP-Anforderung.
- Zwei IBM WebSphere MQ-SOAP-Absender für die Erstellung von IBM WebSphere 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 WebSphere MQ-Transports für SOAP.
- Integration des Clients in Axis2 (gelegentlich auch als 'JAX-WS' bezeichnet) für das Senden von Nachrichten des WebSphere 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 WebSphere MQ-Transports für SOAP zu implementieren.
- Java-und .NET-Beispielclient-und -Servicecode.
- 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 WebSphere MQ

Der WebSphere MQ-Transport für SOAP ist eine Erweiterung von SOAP sowie der Web-Service-Tools und Laufzeit mit WebSphere MQ als Transportalternative zu HTTP für SOAP. Sie müssen bestehende Web-Services nicht für die Verwendung des WebSphere 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 WebSphere MQ-SOAP-Listener führt Services in den Umgebungen mit .NET Framework 1, .NET Framework 2 und Axis 1.4 aus. Der WebSphere MQ-Transport für SOAP ist in einige andere Anwendungsserverumgebungen integriert, darunter WCF, CICS und WebSphere Application Server.

Was ist SOAP?

SOAP[9]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 Version 1.2 prägnant wie folgt:

SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.[10].

Damit SOAP verwendet werden kann, muss es an einen Transport wie HTTP, E-Mail oder WebSphere 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). Die SOAP-HTTP-Bindung wird unter SOAP Version 1.2 Part 2: Adjuncts (Second Edition) beschrieben.

Die W3C -Kandidatenempfehlung, 4. Juni 2009, SOAP over Java Message Service 1.0, beschreibt die Empfehlung für die SOAP JMS-Bindung. 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. WebSphere MQ-JMS erfüllt die JCA-Spezifikation und kann als Plug-in in einem konformen Anwendungsserver verwendet werden.

Der WebSphere 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 erbringt, auf die mithilfe von Internetprotokollen zugegriffen werden kann.

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,

[9] Früher stand das Akronym für Simple Object Access Protocol.
[10] W3C: SOAP Version 1.2 Teil 0

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.

Tipp: Wenn Sie einen Web-Service für die Verwendung von WebSphere 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 WebSphere 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 für Java EE -Entwickler
Erstellt Web-Services für Axis 2. Unterstützt JAX-RPC und JAX-WS

Rational Application Developer V7.5
Erstellt Web-Services für WebSphere Application Server V7 und Vorgängerversionen sowie für Axis. Unterstützt JAX-RPC und JAX-WS.

WebSphere Integration Developer V6.2
Erstellt Web-Services für WebSphere Process Server und WebSphere ESB. Unterstützt JAX-RPC und JAX-WS.

Visual Studio 2008 (Version 9)
Erstellt Web-Services für .NET Framework 3.5 und früher (Windows Communication Foundation).

Visual Studio 2005 (Version 8)
Erstellt Web-Services für .NET Framework 2 und früher

Sie können jedes dieser Tools in Kombination mit dem WebSphere 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 WebSphere 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 WebSphere MQ-Transport für SOAP verwenden.

Wenn der WebSphere 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 WebSphere MQ-Transport für SOAP. Die Server-SOAP-Schicht ruft den WebSphere MQ-Transport für SOAP auf, um auf `jms:`-SOAP-Anforderungen zu warten, und gibt Antworten an den WebSphere-Transport für SOAP zurück.

Normalerweise wurden .NET-Services von unten nach oben unter Verwendung von Web-Service-Annotationen im Code und Java-Services von oben nach unten unter Verwendung von WSDL-Schnittstellendefinitionen entwickelt. Der Unterschied bei den Ansätzen ist, dass Java Standard Edition Version 6 JAX-WS 2.0unterstützt und Annotationen verwendet, um die Definition von Serviceschnittstellen zu qualifizieren. Es ist jetzt so einfach, Java-Services von unten nach oben zu entwickeln wie von oben nach unten. Das von Ihnen gewählte Konzept ist eine Frage des Entwicklungsverfahrens.

Der Web-Service-Client wird nach dem Service geschrieben. Dabei 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 das Stellen von Serviceanforderungen verwendet werden können. 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 vom Client benötigt werden, um Serviceanforderungen zu stellen.

Sie können jede dieser drei Methoden zusammen mit dem WebSphere MQ-Transport für SOAP verwenden.

Web-Service-Anwendungsumgebungen

Web-Service-Toolkits benötigen eine Zuordnung von der WSDL-Definition eines Service zu den Bytetströmen, 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 166 auf Seite 1010 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 166. 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, sind Java-Web-Service-Clients und -Services zwischen verschiedenen Java-basierten Anwendungsumgebungen portierbar. 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. WebSphere MQ 7.0.1 unterstützt JAX-WS und Axis 2 nicht.

Der WebSphere 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 WebSphere MQ-Transport für SOAP aus. WebSphere 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 WebSphere MQ-Transport für SOAP ausgeliefert werden. Die Unterstützung des WebSphere-Transports für SOAP in .NET Framework 3 und 3.5 wird mit dem angepassten WebSphere MQ-Kanal für Windows Communication Foundation implementiert.

Andere SOAP-Entwicklungs- und -Laufzeitumgebungen beinhalten möglicherweise eine Unterstützung des WebSphere 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 WebSphere MQ-Transport für SOAP?

Der WebSphere MQ-Transport für SOAP ist eine SOAP-Bindung und ein Web-Service-Toolkit. Gemeinsam ermöglichen sie Anwendungen den Austausch von SOAP-Nachrichten mit WebSphere MQ statt HTTP.Abbildung 167 auf Seite 1011 zeigt WebSphere MQ als Alternative zu HTTP als SOAP-Transport.

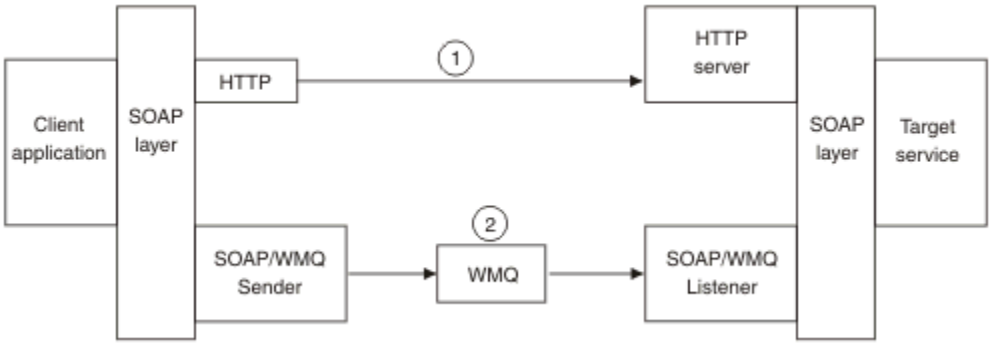


Abbildung 167. Überblick über WebSphere MQ-Transport für SOAP

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 über WebSphere MQ wird angezeigt als (2). Die Clientanwendung registriert die WebSphere MQ-SOAP-Senderkomponente als Handler für das `jms:`-Protokoll mit der SOAP-Schicht. Die SOAP-Schicht übergibt an `jms:` adressierte SOAP-Nachrichten an den WebSphere 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 WebSphere MQ-SOAP-Listener wartet in seiner Anforderungswarteschlange auf Nachrichten und ruft die SOAP-Schicht auf, um Anforderungen zu verarbeiten und Antworten zurückzugeben.

Der SOAP-Absender und -Listener sind normale WebSphere MQ-Programme. Sie können wie in Abbildung 168 auf Seite 1012 mit demselben Warteschlangenmanager oder wie in Abbildung 169 auf Seite 1013 mit unterschiedlichen Warteschlangenmanagern verbunden werden. Der Client kann über eine Clientverbindung verbunden werden.

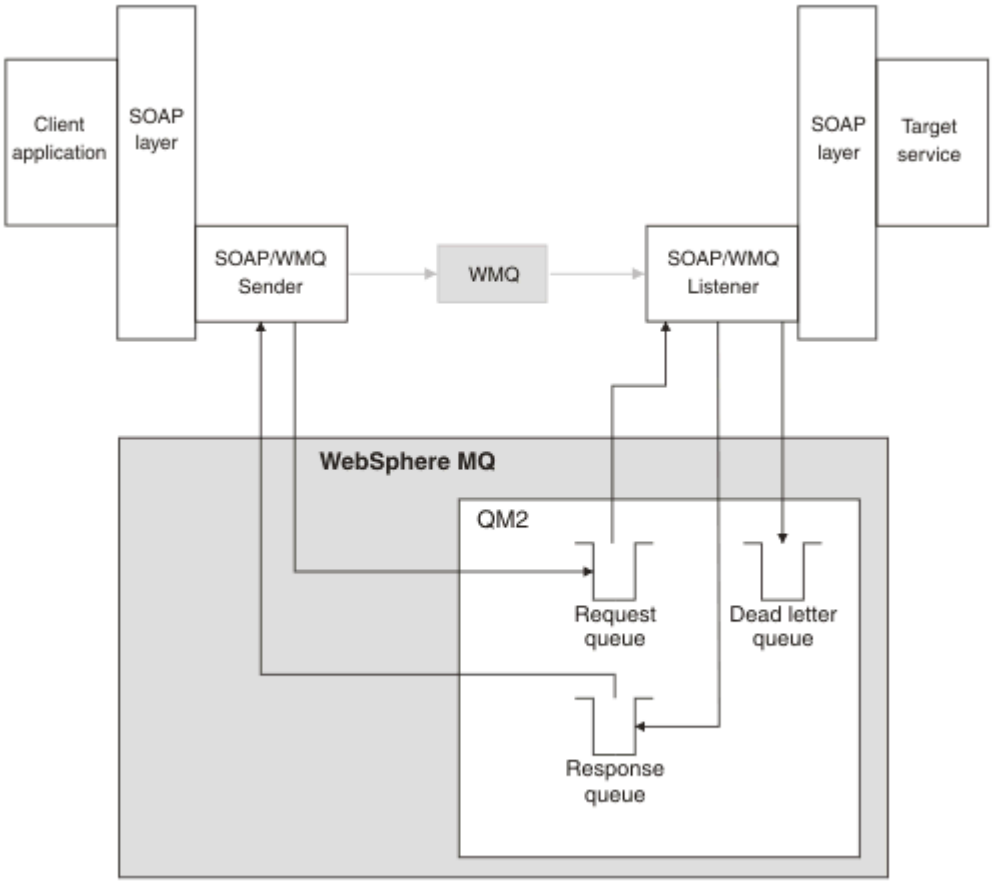


Abbildung 168. Von SOAP/WebSphere MQ verwendete Warteschlangen (einzelner Warteschlangenmanager)

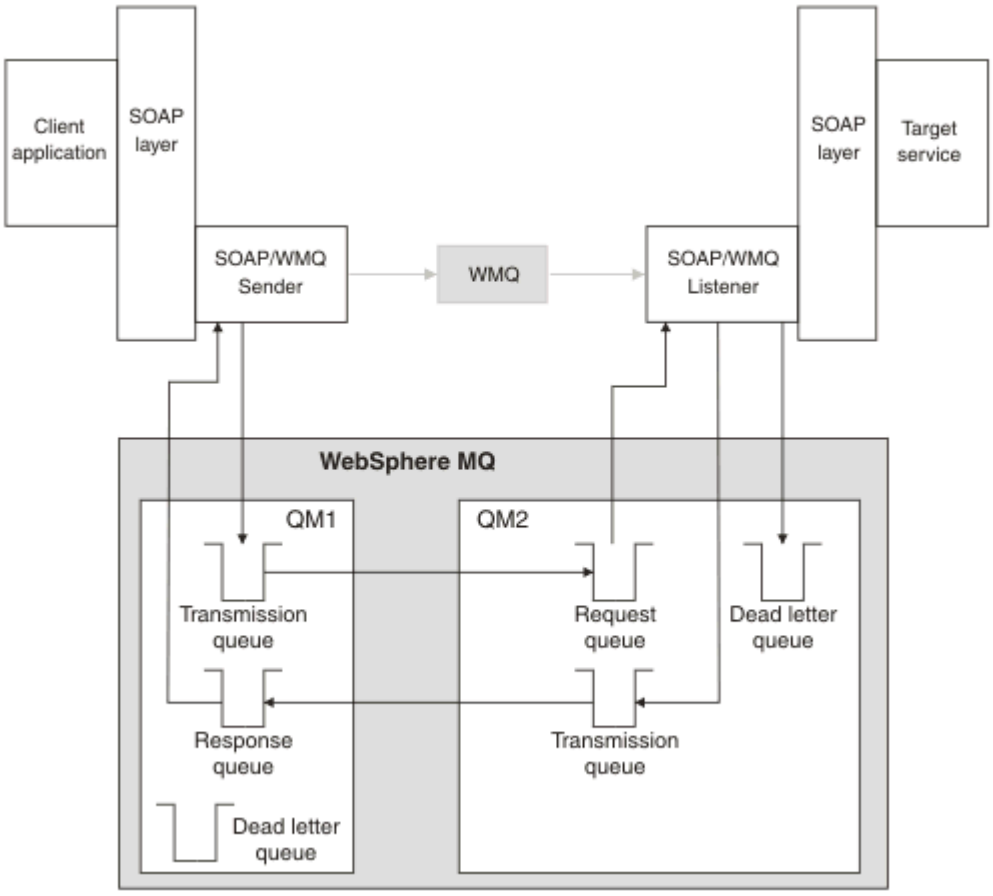


Abbildung 169. Von SOAP/WebSphere MQ verwendete Warteschlangen (separate Warteschlangenmanager)

W3C-Kandidatempfehlung 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 .[11].

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 WebSphere MQ Axis 2-Client. Der Axis2-Client sendet eine SOAP-Anforderung, die entweder mit einem W3C- oder einem WebSphere MQ-Transport für SOAP formatiert ist, auf Basis eines URI in der SOAP-Anforderung.

Die Axis2-Clientunterstützung für die W3C-Empfehlung steht ab dem Fixpack 7.0.1.3 zur Verfügung. Es wird keine Unterstützung für sonstige Clients und die von WebSphere 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 WebSphere MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des WebSphere MQ-Transports für SOAP in .NET Framework 1, .NET Framework 2 und Axis 1.4 als Leitfaden.

WebSphere MQ-Transport für SOAP und Web Services Reliable Messaging

[11] Den aktuellen Entwurf finden Sie in den W3C-Spezifikationsreferenzen unter *URI Scheme for JMS*.

Web Services Reliable Messaging (WS-Reliable Messaging) 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.

Implementierung des WebSphere-Transports für SOAP in .NET Framework 1, .NET 2 und Axis 1.4

Möglicherweise möchten Sie Ihren eigenen WebSphere MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des WebSphere 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 mit 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 WebSphere MQ-Service wird über einen URI angegeben, der das Präfix `jms:` hat. Sobald das Framework den `jms:`-URI ermittelt, ruft es den WebSphere MQ-Transportsendercode auf: `com.ibm.mq.soap.transport.jms.WMQSender` (bei Axis 1.4) oder `IBM.WMQSOAP.MQWebRequest` (bei .NET 1 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 WebSphere MQ-SOAP-Sender transportiert. **`SimpleJavaListener`** (für Java) oder **`amqwSOAPNETListener`** (für .NET) empfängt die Anforderungsnachricht.

   Die WebSphere MQ-SOAP-Listener sind eigenständige Prozesse und werden als Multithread-Prozesse mit einer anpassbaren Anzahl von Threads ausgeführt.
5. Der WebSphere 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 WebSphere MQ-SOAP-Listener zurück.
8. Der Listener stellt die Nachricht in die Antwortwarteschlange und die Nachricht wird an den WebSphere 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 WebSphere MQ-Anforderungswarteschlange bedient.

Der Anwendungskontext wird in Axis 1.4 gesteuert, indem sichergestellt wird, dass der WebSphere 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 WebSphere 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 WebSphere MQ

WebSphere MQ-Transport für SOAP und Web Services Reliable Messaging
Web Services Reliable Messaging (WS-Reliable Messaging) 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.

WebSphere MQ-Transport für SOAP und Web Services Reliable Messaging

Web Services Reliable Messaging (WS-Reliable Messaging) 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.

WebSphere MQ für SOAP nutzt die Vorteile des verwalteten und zuverlässigen WebSphere 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. Sein 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: WebSphere MQ-Transport für 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 WebSphere MQ

Implementierung des WebSphere-Transports für SOAP in .NET Framework 1, .NET 2 und Axis 1.4

Möglicherweise möchten Sie Ihren eigenen WebSphere MQ-SOAP-Sender und -Listener schreiben. Verwenden Sie hierfür die Implementierung des WebSphere MQ-Transports für SOAP in .NET Framework 1, .NET Framework 2 und Axis 1.4 als Leitfaden.

WebSphere MQ-Web-Services installieren und überprüfen

Mithilfe der Anweisungen in diesen Abschnitten können Sie den WebSphere MQ-Transport für SOAP installieren und überprüfen.

WebSphere MQ-Webtransport für SOAP installieren

Mithilfe dieser Anweisungen können Sie den WebSphere MQ-Webtransport für SOAP installieren. Die Installation erstellt Tools für die Ausführung der Web-Service-Clients oder Services, die WebSphere 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 WebSphere MQ. Der Installationsprozess überprüft weder das Vorhandensein noch die Verfügbarkeit der Softwarevoraussetzungen. Sie müssen überprüfen, ob die Voraussetzungen installiert sind.

WebSphere MQ stellt eine Kopie der Laufzeit von Axis 1.4 zur Verfügung. Verwenden Sie diese Version zusammen mit WebSphere 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.

Zur Ausführung von Web-Services in der SOAP-Umgebung mit .NET Framework 3 verwendet WebSphere MQ Windows Communication Foundation. Der angepasste WebSphere MQ-Kanal für Windows Communication Foundation führt Web-Service-Clients und Services unter Verwendung von WebSphere MQ als Transport für SOAP-Nachrichten aus.

Informationen zu diesem Vorgang

Sie können den WebSphere MQ-Webtransport für SOAP entweder als WebSphere MQ-MQI-Client oder als Serveranwendung installieren. Wenn Sie WebSphere 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 WebSphere MQ installiert ist.

Führen Sie die folgenden Installationsschritte aus.

Vorgehensweise

1. Wählen Sie die Komponente "Java and. Net Messaging and Web services" zur Installation aus.
2. Wählen Sie unter Solaris und HP-UXdie Komponente "Java Runtime Environment" für die Installation aus.
3. Wählen Sie das Entwicklungstoolkit für die Installation aus.
4. Installieren und überprüfen Sie WebSphere 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 WebSphere MQ. Kopieren Sie sie in das Installationsverzeichnis, das in Tabelle 138 auf Seite 1017, Tabelle 139 auf Seite 1017 oder Tabelle 140 auf Seite 1017 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 WebSphere MQ verweist.

Ergebnisse

Tabelle 138. Windows-Installationsverzeichnisse

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 139. AIX-Installationsverzeichnisse

Position	Inhalt
MQ_INSTALLATION_PATH/bin	Shell-Scripts
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar und andere JAX-RPC- .jar -Dateien
MQ_INSTALLATION_PATH/samp/soap	Beispiele und Installationsprüftest

Tabelle 140. 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 andere JAX-RPC- .jar -Dateien
MQ_INSTALLATION_PATH/samp/soap	Beispiele und Installationsprüftest

Nächste Schritte

1. Gilt nur für .NET: Sie müssen die Dateien des WebSphere MQ-Transports für SOAP beim Global Assembly Cache registrieren. Wenn .NET bei der Installation von WebSphere MQ bereits installiert ist, wird die Registrierung bei der Installation automatisch ausgeführt. Wenn Sie .NET nach WebSphere 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 WebSphere MQ-Transport für SOAP überprüfen" auf Seite 1018 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 unter Verwendung von Eclipse entwickeln" auf Seite 1041.

IBM WebSphere MQ-Transport für SOAP überprüfen

Die Installation des IBM WebSphere 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.

Vorbereitende Schritte

Stellen Sie vor Ausführung des Befehls **runivt** sicher, dass Sie über folgende Laufzeitumgebungen verfügen:

- Nur zur Ausführung auf Axis: Auf Ihrem System muss ein Java SDK (in SOE) verfügbar sein. Außerdem müssen Sie der Umgebungsvariablen **PATH** des Systems die Speicherposition der Befehle `java.exe` und `javac.exe` hinzufügen.
- Führen Sie einen Test nur unter .NET aus (nur unter Windowsunterstützt): Sie müssen sowohl über ein Java SDK als auch über die .NET-Compiler und -Tools auf Ihrem System verfügen. Rufen Sie dazu entweder eine Visual Studio-Eingabeaufforderung oder die Microsoft Windows SDK-Eingabeaufforderung auf und fügen Sie dann die Position der Dateien `java.exe` und `javac.exe` zur Umgebungsvariablen **PATH** 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

Statt die Prüfung für .NET und Axis auszuführen, können Sie sie auch nur für Axis oder nur für .NET ausführen.

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.

So überprüfen Sie die Installation des IBM WebSphere MQ-Transports für SOAP unter .NET Framework 1, .NET Framework 2 und Axis 1.4:

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 für die Testklasse, den Beispielclient und die auszuführenden Services die Installationsschritte aus, die in WebSphere(r) MQ Web Transport for SOAP installieren beschrieben sind, und stellen Sie sicher, dass in der Eingabeaufforderung, die zum Ausführen des Befehls `runivt` verwendet wird, die erforderliche Laufzeitumgebung festgelegt ist. 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 WebSphere 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 Verweise

runivt: Installationsprüftest für WebSphere MQ-Transport für SOAP

Web-Services für WebSphere MQ-Transport für SOAP entwickeln

Zur Entwicklung von Services für WebSphere MQ-Transport für SOAP verwenden Sie Ihre ganz normale Entwicklungsumgebung für Web-Services.

Vorbereitende Schritte

1. Wenn Sie die mit WebSphere MQ-Transport für 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. Die Beispielentwicklungstasks verwenden Microsoft Visual Studio 2008, Eclipse IDE für Java EE -Entwickler und WebSphere Application Server Community Edition.

Informationen zu diesem Vorgang

An den vorhandenen Web-Services müssen zur Unterstützung von WebSphere-Transport für SOAP keine Änderungen vorgenommen werden. Mit den mit WebSphere MQ-Transport für SOAP bereitgestellten Tools kann ein Web-Service implementiert und mit einem WebSphere MQ SOAP-Empfangsprogramm ausgeführt werden. Außerdem können mit diesen Tools WSDL, .NET-Client-Stubs und `.java`-Proxy-Klassen für die Entwicklung von WebSphere MQ-Transport für 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 in einem HTTP-Web-Services-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. Stellen Sie sicher, dass der Klassenpfad alle erforderlichen Klassen finden kann: Kompilieren Sie 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` .

JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln

Entwicklung eines Axis 1.4-Web-Service zur Verwendung von WebSphere MQ als Service-Provider. Verwenden Sie Ihre normale Web-Service-Entwicklungsumgebung, um einen Service für die Implementierung in Axis 1.4. zu erstellen.

Vorbereitende Schritte

Berücksichtigen Sie die Anforderungen zum Bereitstellen eines Web-Servers für den WebSphere MQ SOAP-Listener für Axis 1.4.

- Der WebSphere MQ SOAP-Listener für Axis 1.4 benötigt mindestens die IBM JRE Version 5.0. Die bei der Entwicklung verwendete JRE und das JDK müssen denselben Versionsstand aufweisen.
- Der WebSphere MQ SOAP-Listener für Axis 1.4 erfordert, dass die Datei `axis.jar` mit WebSphere MQ installiert ist. Ändern Sie den Buildpfad in Ihrer Entwicklungsumgebung so, dass er auf die Datei `axis.jar` verweist, die mit WebSphere MQinstalliert wurde, und nicht auf die `axis.jar` -Dateien, die mit der Entwicklungsumgebung installiert wurden.
- Bis einschließlich WebSphere MQ V7.0.1 ist die für den implementierten Service generierte WSDL RPC/-codiert. Ab Version 7.1 können Sie auch eine WSDL vom Typ RPC/Literal anfordern. Die generierte WSDL wird nur zur Implementierung verwendet. Sie können den Service mit einer WS-I-konformen WSDL definieren.

Informationen zu diesem Vorgang

Erstellen Sie den Service mit Ihrer normalen Web-Service-Entwicklungsumgebung.

In dieser Task verwenden wir die frei verfügbare Open-Source-IDE Eclipse Java EE für Webentwickler, bekannt als Galileo. Für den Anwendungsserver verwenden wir WebSphere Community Edition v2.1 (Community Edition), basierend auf Geronimo. Informationen zum Anfordern, Installieren und Konfigurieren der IDE und des Servers finden Sie in den entsprechenden Tasks.

Testen Sie den Service mit HTTP als Transport mit dem Web Services Explorer der IDE. Generieren Sie alternativ einen HTTP-Client-Proxy und testen Sie den Service mit Ihrem eigenen Client-Code.

Führen Sie diese Schritte aus, wenn Sie einen Bottom-up-Web-Service entwickeln möchten. Verwenden Sie das Beispielprogramm `StockQuoteAxis.java` als Beispiel.

Vorgehensweise

1. Starten Sie Eclipse IDE für Java EE -Entwickler mit einem neuen Arbeitsbereich.
2. Konfigurieren Sie den Arbeitsbereich für die Verwendung von Java50,
   WebSphere Application Server Community Edition 2.1.4 kann nicht mit Java60 ausgeführt werden.
   a) **Fenster** > **Vorgaben** > **Java** > **Installierte JREs** > **Hinzufügen ...** > **Standard-VM** > **Weiter** > **Verzeichnis**
   b) Navigieren Sie zum Installationsverzeichnis von **Java50** > **OK** > **Fertigstellen** .
   c) Überprüfen Sie **Java50** JRE > OK
3. Fügen Sie die Community Edition-Laufzeitumgebung hinzu und starten Sie die Community Edition.
   a) **Fenster** > **Benutzervorgaben** > **Server** > **Laufzeitumgebungen** > **Hinzufügen**
   b) Wählen Sie **IBM WASCE v2.1** in der Liste **Neue Serverlaufzeitumgebung** aus und wählen Sie **Neuen lokalen Server erstellen** > **Weiter** aus.
      Wenn **IBM WASCE v2.1** nicht in der Liste aufgeführt ist, müssen Sie zwei andere Tasks ausführen:
      i) Installieren Sie WebSphere Application Server Community Edition.
      ii) Installieren Sie das Eclipse-Update für die Community Edition.
      Ausführliche Informationen hierzu finden Sie unter WebSphere Application Server Community Edition.
   c) Navigieren Sie zum Installationsverzeichnis des Anwendungsservers > **OK** > **Fertigstellen** > **OK**.
   d) Klicken Sie in der Ansicht "Server" mit der rechten Maustaste auf **IBM WASCE v2.1 Server** > **Starten**
      Tipp: Sie können WASCE in Eclipseverwalten: Klicken Sie mit der rechten Maustaste auf **IBM WASCE v2.1 Server** > **WASCE-Konsole starten** . Die Standardwerte für **`Username`** und **`Password`** sind `system` und `manager` .
4. Richten Sie den Server und die Laufzeit für Web-Services ein.
   a) **Fenster** > **Benutzervorgaben** > **Web-Services** > **Server und Laufzeit**
   b) Wählen Sie **IBM WASCE v2.1 Server** als Server aus.
   c) Übernehmen Sie **Apache Axis** als Web-Service-Laufzeit.
5. Erstellen Sie ein dynamisches Webprojekt.
   a) **Datei** > **Neu** > **Dynamisches Webprojekt**.
      Benennen Sie das Projekt `StockQuoteAxis`.
   b) Wählen Sie **Projekt zu einer EAR-Datei hinzufügen** > **Neu** aus.
   c) Geben Sie auf der Seite **EAR-Anwendungsprojekt** Folgendes ein: **`Project name`** `StockQuoteAxisEAR` > **Fertigstellen**
      Antworten Sie mit **OK** als Antwort auf das Dialogfenster, in dem Sie vorschlagen, zur Java EE -Perspektive zu wechseln, oder bleiben Sie in der Java-Perspektive, um genau diesen Anweisungen zu folgen.
   d) **IBM WASCE 2.1-Server** ist als Ziellaufzeit ausgewählt. Akzeptieren Sie sie und die anderen Standardwerte > **Fertigstellen**.

Antworten Sie mit **OK** als Antwort auf das Dialogfenster, in dem Sie vorschlagen, zur Java EE -Perspektive zu wechseln, oder bleiben Sie in der Java-Perspektive, um genau diesen Anweisungen zu folgen.

6. Importieren Sie das Beispielprogramm `StockQuoteAxis.java`.
   a) Öffnen Sie das Webprojekt **StockQuoteAxis** und klicken Sie mit der rechten Maustaste auf den Ordner **`src`** > **Importieren ...** .
   b) Wählen Sie **Allgemein** > **Dateisystem** > **Weiter** aus.
   c) Navigieren Sie zu *`MQ_INSTALLATION_PATH`*`\tools\soap\samples\java\server` > check **`StockQuoteAxis.java`** > **Finish** .

      `MQ_INSTALLATION_PATH` ist dabei immer das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist. Damit die darin enthaltenen Dateien angezeigt werden, müssen Sie das Serververzeichnis markieren.
7. Korrigieren Sie den Kompilierungsfehler, indem Sie `StockQuoteAxis.java` in das richtige Paket verschieben.
   a) Öffnen Sie `StockQuoteAxis.java` und klicken Sie mit der rechten Maustaste auf das Problem > **Schnellkorrektur** .
   b) Klicken Sie doppelt auf **Move 'StockQuoteAxis.java' to package 'soap.server'** > **Speichern**.
8. Erstellen Sie einen Web-Service mit `StockQuoteAxis.java`.
   a) Klicken Sie mit der rechten Maustaste auf `StockQuoteAxis.java` > **Web-Services** > **Web-Service erstellen** > **Weiter**.

      Übernehmen Sie die Standardkonfiguration für den Service:

      Web-Service-Typ
      Bottom-up-Java- beanWeb -Service

      Serviceimplementierung
      soap.server.StockQuoteAxis

      Server
      `IBM WASCE v2.1 server`

      Web-Service-Laufzeit
      Apache Axis

      Serviceprojekt
      StockQuoteAxis

      Service-EAR-Projekt
      StockQuoteAxisEAR

      Konfiguration
      Keine Client-Generierung
9. Wählen Sie die Methoden für den Zugriff und den Stil des Web-Service aus > **Weiter**.

   Starten Sie den Server, wenn Sie dazu aufgefordert werden.
   a) Lassen Sie alle Methoden ausgewählt.
   b) Wählen Sie den Stil Dokument/Literal (wrapped) aus.
10. **Fertig stellen**

    Nachdem der Service implementiert wurde, suchen Sie im Ordner `WebContent\wsdl` im Web-Projekt `StockQuoteAxis` nach der generierten Datei `StockQuoteAxis.wsdl`.
11. Testen Sie den Service mit HTTP mit dem Web Services Explorer.
    a) Klicken Sie mit der rechten Maustaste auf `StockQuoteAxis.wsdl` > **Mit Web-Services-Explorer testen**.
    b) Klicken Sie im Fenster **Web Services Explorer** in den Aktionen **StockQuoteAxisSoapBinding** auf die Operation **getQuote**.
    c) Geben Sie `ibm` in das Eingabefeld **Symbol** ein > **Los** .
12. Testen Sie den Service mit WebSphere MQ-Transport für SOAP.

Verwenden Sie **SimpleJavaListener**in `com.ibm.mq.soap.jar`, um den Service zu implementieren. Sie müssen die Java-und SOAP-Bibliotheken WebSphere MQ zum Buildpfad hinzufügen.

a) Klicken Sie mit der rechten Maustaste auf **StockQuoteAxis** Webprojekt > **Buildpfad** > **Buildpfad konfigurieren ...**

b) Klicken Sie auf die Registerkarte **Bibliotheken** > **Externe JAR-Dateien hinzufügen ...**. Navigieren Sie zu *MQ_INSTALLATION_PATH*`\java\lib` und wählen Sie alle `.jar` -Dateien > **Öffnen** > **Externe JAR-Dateien hinzufügen ...**aus. Navigieren Sie zu *WMQ Install directory*`\java\lib\soap` und wählen Sie alle `.jar` -Dateien > **Öffnen** > **OK**aus.

MQ_INSTALLATION_PATH ist dabei immer das übergeordnete Verzeichnis, in dem WebSphere MQ installiert ist.

c) Klicken Sie im Projektexplorer mit der rechten Maustaste auf `StockQuoteAxis\Java Resources\Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class\ SimpleJavaListener` > **Ausführen als ...**> **Ausführungskonfigurationen ...** .

Tipp:

Ist keine Konfiguration für `SimpleJavaListener` vorhanden, klicken Sie im Assistenten zum **Ausführen von Konfigurationen** auf der Seite **Konfigurationen erstellen, verwalten und ausführen** auf das Symbol **Neue Konfiguration**.

Es gibt keinen Befehl zum Stoppen des **SimpleJavaListener**. Um **SimpleJavaListener** zu überwachen oder zu stoppen, öffnen Sie in Eclipse die **Debugperspektive**.

d) Öffnen Sie die Registerkarte **(x)= Argumente**. Geben Sie im Eingabebereich **Programmargumente** die Parameter für **SimpleJavaListener** ein.

Geben Sie in diesem Beispiel Folgendes ein:

```
-u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
```

Anmerkung: Der Zielservice `StockQuoteAxis` muss mit dem Namen des Zielservice übereinstimmen, der im Serviceimplementierungsdeskriptor `StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd` erstellt wird. amqwdeployWMQService erstellt einen Zielservice mit dem Namen `soap.server.StockQuoteAxis`. In diesem Beispiel verwenden Sie dieselben `StockQuoteAxis.class`- und `service-config.wsdd`-Dateien wie der HTTP-Server.

e) Konfigurieren Sie auf derselben Registerkarte das **Arbeitsverzeichnis** so, dass es auf die Datei `server-config.wsdd` verweist:
`${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}`

a) **Ausführen**

Fehler werden in die Konsole geschrieben. Bleibt die Konsole leer, wurde **SimpleJavaListener** erfolgreich gestartet.

b) Testen Sie die Implementierung, indem Sie den in der Task „JAX-RPC-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln” auf Seite 1033 entwickelten Client 'StockQuoteAxis' ausführen.

Beispiel: Beispielprogramm 'StockQuoteAxis'

Der Java-Beispiel-Web-Service `StockQuoteAxis.java`ist in *WMQ install directory*`\tools\soap\samples\java\server` installiert. `StockQuoteAxis.java`, Abbildung 170 auf Seite 1024, hat vier Methoden:

1. `float getQuote(String symbol)`
2. `void getQuoteOneWay(String symbol).`
3. `int asyncQuote(int delay)`
4. `float getQuoteTran(String symbol)`

```
package soap.server;
import java.lang.Thread;
import java.io.FileWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }

    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,
                exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

Abbildung 170. StockQuoteAxis

Nächste Schritte

Implementieren Sie den Service mit WebSphere MQ-Transport für SOAP anstatt mit HTTP, indem Sie den Befehl **amqwdeployWMQService** verwenden.

Die Option `axisDeploy` dieses Befehls implementiert den Service, indem ein Apache Axis 1.4-Implementierungsdeskriptor erstellt wird. Der WebSphere MQ SOAP-Listener führt den Service aus. Der SOAP-Listener heißt SimpleJavaListener und wird mit dem WebSphere MQ -Transport für SOAP bereitgestellt.

Zugehörige Tasks

.NET 1- oder 2-Service für den WebSphere MQ-Transport für SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln
Entwicklung des Web-Service 'SampleStockQuote' für .NET 1 oder .NET 2 mithilfe von Microsoft Visual Studio 2008

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.

.NET 1- oder 2-Service für den WebSphere MQ-Transport für SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln

Entwicklung des Web-Service 'SampleStockQuote' für .NET 1 oder .NET 2 mithilfe von Microsoft Visual Studio 2008

Informationen zu diesem Vorgang

Erstellen Sie unter Verwendung von Visual Studio 2008 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 2008 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 171 auf Seite 1026.*MQ_INSTALLATION_PATH* steht für das Verzeichnis, in dem WebSphere 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 WebSphere MQ-Transport für 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 WebSphere MQ-Transports für 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* Das Verzeichnis, in dem WebSphere MQ installiert ist. Die Web-Service-Bindung der veröffentlichten Beispiele weicht minimal von der in der Task verwendeten Bindung ab. Die Task verwendet die Standardeinstellungen und -werte von Visual Studio 2008.

Es gibt zwei Beispiele für .NET Framework 1 und .NET Framework 2 Web-Services. `StockQuoteDotNet.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 171. Integrierter Service: `StockQuoteDotNet.asmx`

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Abbildung 172. 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 173. Code-Behind: Implementierung: SQDNNonInline.asmx.cs

Zugehörige Tasks

JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis 1.4-Web-Service zur Verwendung von WebSphere MQ als Service-Provider. Verwenden Sie Ihre normale Web-Service-Entwicklungsumgebung, um einen Service für die Implementierung in Axis 1.4. zu erstellen.

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. Führen Sie „WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren" auf Seite 1067 aus.

Um die in der Task verwendete WSDL zu erstellen, müssen Sie zunächst die Task „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln" auf Seite 1020 ausführen. Danach können Sie die WSDL entweder aus dem dynamischen Webprojekt im Eclipse Galileo-Arbeitsbereich oder aus dem aktiven, in WASCE implementierten HTTP-Web-Service importieren.

WebSphere Application Server wird unter Umständen weiterhin als Ergebnis von Schritt „WebSphere Application Server-Ressourcen konfigurieren" auf Seite 1068 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 WebSphere 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 WebSphere 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 prüfen Sie, ob WebSphere Application Server läuft.
   a) Starten Sie Rational Application Developer in einem neuen Arbeitsbereich.
   b) Öffnen Sie die Perspektive Java EE .
   c) Öffnen Sie die Registerkarte **Server** und prüfen Sie, ob WebSphere Application Server läuft.
      - Wenn WebSphere Application Server v7.0 nicht in der Ansicht vorhanden ist, klicken Sie mit der rechten Maustaste in die Ansicht > **Neu** > **Server**. Folgen Sie den Anweisungen im Assistenten, um eine Instanz von WebSphere Application Server v7.0 zu erstellen.
      - Ist der Server vorhanden, wurde aber noch nicht gestartet, klicken Sie zum Starten auf die Pfeilspitze.
      - Um die Eigenschaften zu überprüfen und schnell auf die Serverprotokolle zuzugreifen, klicken Sie mit der rechten Maustaste auf **WebSphere Application Server v7.0 unter '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 v7.0 unter localhost** > **Administrationskonsole ausführen**.
- Standardmäßig werden Veröffentlichungen automatisch ausgeführt. Viele Benutzer ziehen es vor, Server-Updates manuell zu implementieren. Doppelklicken Sie auf **WebSphere Application Server v7.0 im lokalen Host** und erweitern Sie **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 Project (EAR) und ein EJB-Projekt erstellen.

   a) Klicken Sie auf **Datei** > **Neu** > **EAR-Projekt**. Benennen Sie das Projekt `W3CJMSEAR` > **Fertigstellen**.

   Die Standardeinstellungen müssen WebSphere Application Server v7.0 als Ziellaufzeit und die EAR-Version 5.0 angeben. 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) Wählen Sie das Kontrollkästchen **JAR-Modul für EJB-Client erstellen** ab und klicken Sie auf **Fertigstellen**.

3. Generieren und implementieren Sie den EJB-Web-Service in der WSDL `StockQuoteAxis`.

   a) Klicken Sie auf **Ausführen** > **Web Services Explorer starten**.

   b) Wählen Sie die WSDL-Seite mithilfe der Symbole im Fenster**Web-Service-Explorer** aus und klicken Sie im Navigatorauf **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 WASCE 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 141 auf Seite 1029. Wählen Sie **Dateien ohne Warnung überschreiben** > **Weiter** aus.

Tabelle 141. Top down-Konfiguration des EJB-Web-Service

Feld	**Wert**
Server	`WebSphere Application Server v7.0`
Web-Service-Laufzeit	`IBM WebSphere JAX-WS`
Serviceprojekt	`W3CJMSEJB`

Tabelle 141. Top down-Konfiguration des EJB-Web-Service (Forts.)	
Feld	**Wert**
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**. Wählen Sie außerdem **Wrapper-Stil aktivieren**, **WSDL in Projekt kopieren** und **Web-Service-Implementierungsdeskriptor generieren** > **Weiter**aus.

h) Wählen Sie auf der Seite mit dem Titel **WebSphere JAX-WS-JMS-Bindungskonfiguration**die Option **SOAP/JMS-Interoperabilitätsprotokoll verwenden** aus und geben Sie Werte aus Tabelle 142 auf Seite 1030an. Lassen Sie die anderen Felder leer > **Weiter**.

Tabelle 142. Konfiguration der WebSphere-JAX-WS-JMS-Bindung	
Feld	**Wert**
JMS-Zieladresse	`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** > **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.

Indem Sie der EJB zwei Bindungen bereitstellen, können Clients die SOAP-Bindungen zum Aufrufen des Web-Service auswählen. 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 1078

Zugehörige Tasks

JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln
Entwicklung eines Axis 1.4-Web-Service zur Verwendung von WebSphere MQ als Service-Provider. Verwenden Sie Ihre normale Web-Service-Entwicklungsumgebung, um einen Service für die Implementierung in Axis 1.4. zu erstellen.

.NET 1- oder 2-Service für den WebSphere MQ-Transport für SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln
Entwicklung des Web-Service 'SampleStockQuote' für .NET 1 oder .NET 2 mithilfe von Microsoft Visual Studio 2008

WebSphere MQ-Web-Service-Clients für WebSphere MQ Transport for SOAP entwickeln

Mit Ihrer normalen Entwicklungsumgebung entwickeln Sie Web-Service-Clients zur Verwendung mit WebSphere MQ Transport for SOAP.

Vorbereitende Schritte

Erstellen Sie den Service. Dazu können Sie eines der Beispielprogramme in „Web-Services für WebSphere MQ-Transport für SOAP entwickeln” auf Seite 1019 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 WebSphere MQ Transport for SOAP entwickeln möchten.

Es gibt zwei Methoden.

1. Sie verwenden die Standardentwicklungstools, 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 WebSphere MQ Transport for SOAP.

Wenn Sie die HTTP-Methode verwenden, können Sie den Service sowohl auf einem HTTP-Server als auch mit WebSphere MQ Transport for SOAP ausführen. Zur Ausführung mit WebSphere MQ Transport for SOAP konfigurieren Sie das entsprechende WebSphere MQ-Empfangsprogramm für SOAP

d>-Listener für SOAP und legen die Pfade und Implementierungsdeskriptoren für die Ausführung des Service fest. Die Tools in WebSphere MQ Transport for SOAP führen dann die Konfiguration aus. Alternativ können Sie die Umgebung für die Ausführung der Listener konfigurieren.

Die Tools in WebSphere 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 programmgestützt 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 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 von WebSphere MQ Transport for 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 WebSphere 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. Aus der .NET-Serviceklasse (.asmx) abgerufene WSDL. mit **disco**.
3. WSDL, die mit der URL *Web service http url* ?wsdlabgefragt wird, falls verfügbar. Führen Sie ein Tool wie Web Services Explorer aus, um die Servicedefinition direkt in Eclipse zu importieren.
4. Durch Ausführen von **amqswsdl** für die Serviceklasse .NET (.asmx) 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 WebSphere 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 .NET 1 und 2, Axis 1.4 (JAX-RPC) und Axis2 (JAX-WS) erstellen. Informationen zur Vorgehensweise für Windows Communication Foundation finden Sie unter den zugehörigen Task-Links.

JAX-RPC-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln

Entwickeln Sie einen Axis 1.4-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

Vorbereitende Schritte

Der Service muss verfügbar sein. Wenn Sie die Task als praktische Übung ausführen, verwenden Sie den in der Task „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020 erstellten Arbeitsbereich und Service. Stellen Sie sicher, dass ein Anwendungsserver in Eclipse ausgeführt wird, der Axis 1.4 Web-Services unterstützt. In dieser Task verwenden wir die frei verfügbare WebSphere Application Server Community Edition Version 2.1.4. Sie wird in der Task „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020 konfiguriert. 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 mit dem Namen `StockQuoteAxisClient`:
   a) Wechseln Sie zur Java-Perspektive > **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 **J2SE1-1.4** oder **J2SE-1.5** > **Weiter** ist.
   b) Wählen Sie auf der Seite **Java-Einstellungen** in der Registerkarte **Bibliotheken** die Option **Externe JAR-Dateien hinzufügen** aus.
   c) Navigieren Sie zu *MQ_INSTALLATION_PATH*`/java/lib` und wählen Sie alle `.jar` -Dateien > **Öffnen**aus.

      MQ_INSTALLATION_PATH ist das Verzeichnis, in dem WebSphere MQ installiert ist.
   d) Navigieren Sie zu *MQ_INSTALLATION_PATH*`/java/lib/soap` und wählen Sie alle `.jar` -Dateien > **Öffnen**aus. `axis.jar` muss aus dem WebSphere MQ-Installationsmedium in diesem Verzeichnis installiert werden.

      MQ_INSTALLATION_PATH ist das Verzeichnis, in dem WebSphere 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 Proxys in Eclipse 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 `samples/soap/server`.

     c. Ändern Sie `amqwsetcp.cmd` in *MQ_INSTALLATION_PATH*`/bin`, um das aktuelle Verzeichnis im Klassenpfad einzuschließen.*MQ_INSTALLATION_PATH* Das Verzeichnis, in dem WebSphere 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 `"/"`anstelle von `"."` oder `"\"` , wenn Sie Java-Befehle verwenden.

     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 von Eclipse generierten Proxys enthalten eine 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) Wählen Sie **Server** auf der Seite**Importieren** > **Fertigstellen**aus.

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 steht der Service über die WebSphere Application Server Community Edition zur Verfügung und Sie erhalten die WSDL vom Web-Server. . Die Implementierung wird in der Task „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020 beschrieben,

  a. Wechseln Sie in Eclipse zur Webperspektive und stellen Sie sicher, dass der WebSphere Application Server Community Edition v2.1-Server ausgeführt wird und 'StockQuoteAxis' implementiert 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 in Task „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020, 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 von WebSphere MQ Transport for 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 main-Methode:

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 177 auf Seite 1040 bis Abbildung 181 auf Seite 1041 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 WebSphere 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 177 auf Seite 1040 und Abbildung 179 auf Seite 1040 verwenden den in Eclipse generierten Proxy, der über die zusätzliche Helper-Klasse 'StockQuoteAxisproxy' verfügt, welche das Codieren etwas vereinfacht.

– Abbildung 178 auf Seite 1040 und Abbildung 180 auf Seite 1041 verwenden den mit **amqwdeployWMQService** generierten Proxy.

– Abbildung 181 auf Seite 1041 verwendet keine Proxy-Klassen.

Jeder Client ruft `com.ibm.mq.soap.Register.extension()` auf, um eine Verknüpfung zum WebSphere 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 WebSphere MQ Transport for SOAP implementieren” auf Seite 1073.

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 WebSphere MQ Transport for SOAP aus.

Die Prozedur verwendet **amqwdeployWMQService** , um den Service zu implementieren, und funktioniert nur mit dem Client, der die WSDL oder Proxys verwendet, die von **amqwdeployWMQService** erstellt wurden. 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 im Abschnitt „Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren.” auf Seite 1061 , um den Service in WebSphere 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 Clientimplementierungsdeskriptor `client-deploy.wsdd` zu erstellen.

iii) Importieren Sie `client-deploy.wsdd` in das Stammverzeichnis des Java-Projekts, das Sie mit WebSphere MQ Transport for SOAP testen möchten.

   a) Klicken Sie mit der rechten Maustaste auf das Java-Projekt **StockQuoteAxisEclipseClient** > **Importieren** > **Dateisystem** > **Weiter** > **Durchsuchen ...**

   b) Navigieren Sie zu dem Verzeichnis mit `client-deploy.wsdd` > **Öffnen** > Wählen Sie das Verzeichnis auf der Seite des **Importassistenten** aus und klicken Sie im rechten Teilfenster auf `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) Vergewissern Sie sich, dass das Arbeitsverzeichnis zum Ausführen einer Java-Anwendung in diesem Projekt das Verzeichnis `StockQuoteAxisEclipseClient` ist:

   Klicken Sie mit der rechten Maustaste auf das Java-Projekt **StockQuoteAxisEclipseClient** > **Ausführen als ...** > **Ausführungskonfigurationen ...** > Wählen Sie die Registerkarte **(x) = Argumente** aus > Überprüfen 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 WebSphere 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 Buildpfad-ohne die JAR-Dateien in *`MQ_INSTALLATION_PATH`*`/java/lib` . *`MQ_INSTALLATION_PATH`* ist das Verzeichnis, in dem WebSphere MQ installiert ist.

iii) Problem bei der Serviceimplementierung, 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, die mit WebSphere MQ geliefert werden, sind im Verzeichnis *MQ_INSTALLATION_PATH* `\tools\soap\samples\java\clients`installiert. *MQ_INSTALLATION_PATH* Das Verzeichnis, in dem WebSphere MQ installiert ist.

SQAxis2Axis.java
: `SQAxis2Axis.java`, Abbildung 174 auf Seite 1038, 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 175 auf Seite 1039, 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 176 auf Seite 1039, ist ein dynamischer Aufrufclient zum Aufrufen des Service `StockQuoteDotNet` oder `StockQuoteAxis` . Der Client ruft standardmäßig den Service `StockQuoteAxis` auf. Fügen Sie die Befehlszeilenoption -D hinzu, um den `StockQuoteDotNet` -Service aufzurufen, und -w, um einen anderen Port als den in `.\generated\StockQuoteDotNet_Wmq.wsdl` anzugeben.

```
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 174. 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 175. 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 176. 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 177. 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 178. 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 179. 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 180. 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 181. DII-Client (kein Proxy)

Zugehörige Tasks

JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln
Entwickeln Sie einen Axis2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP. Die Axis2-Musterclients, die im Rahmen von WebSphere MQ Transport for SOAP bereitgestellt werden, werden aufgelistet. Außerdem wird der Befehl **wsimport** erläutert, mit dem Proxys generiert werden.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln
Entwickeln Sie einen .NET 1- oder 2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln

Entwickeln Sie einen Axis2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP. Die Axis2-Musterclients, die im Rahmen von WebSphere MQ Transport for SOAP bereitgestellt werden, werden aufgelistet. Außerdem wird der Befehl **wsimport** erläutert, mit dem Proxys generiert werden.

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 WebSphere MQ ausgeliefert. Verwenden Sie für Axis 1.4 die Version von `axis.jar`, die mit WebSphere MQ ausgeliefert wird, und nicht die 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). Die Releaseversion von Axis2, die für WebSphere MQ Transport for SOAP verwendet wird, ist 1.4.1. Laden Sie Axis2 1.4.1 von der Website http://ws.apache.org/axis2/download/1_4_1/download.cgi herunter.

Sie können für die Generierung von Proxys für die Web-Service-Clients für WebSphere MQ Transport for SOAP entweder **`wsimport`** oder die Tools verwenden, die mit einer integrierten Entwicklungsumgebung bereitgestellt werden. Eclipse IDE for Java EE Entwickler 3.5 SR1 verwendet **`wsdl2java`**.**`wsimport`** wird mit Java 6 bereitgestellt. Sie können Java 5 verwenden, um Client-Proxys auszuführen, die mit **`wsimport`** oder **`wsdl2java`**generiert wurden.

Die Mustercodes der Axis2-Web-Service-Clients, die mit WebSphere MQ Transport for SOAP bereitgestellt werden, wurden mit **`wsimport`** entwickelt; siehe „Axis2-Beispielclients” auf Seite 1047.

Die folgende Task veranschaulicht, wie die vom Web-Service-Assistenten erstellten Proxys generiert und verwendet werden, der mit Eclipse IDE für Java EE -Entwickler gepackt ist. 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 WASCE für die Unterstützung von Axis2 mit der Workbench konfigurieren können.

1. Konfigurieren Sie den Anwendungsserver, der in Eclipse IDE für Java EE -Entwickler verwendet wird, um Axis2zu unterstützen. In diesem Beispiel konfigurieren Sie den Anwendungsserver WASCE 2.1.4, der Bestandteil des in „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020 erstellten Arbeitsbereichs ist.
   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 WASCE als Server hinzu: Klicken Sie auf **Server** > **Laufzeitumgebungen** > **Hinzufügen ...** > **IBM** > **WASCE v2.1** > Weiter. Die JRE muss Java50 > Durchsuchen Sie das WASCE-Installationsverzeichnis > **OK** > **Fertigstellen**sein. Sie müssen das WASCE-Plug-in für Eclipse Java EE IDE für Webentwickler installiert haben.
   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. Ordnen Sie WASCE Axis2: Klicken Sie auf **Web-Services** > **Server und Laufzeit**. Wählen Sie unter **`Server` IBM WASCE v2.1 Server**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. **IBM WASCE v2.1 Server** ist ausgewählt und konfiguriert > **Fertigstellen**. Starten Sie den-Server.

2. Vergewissern Sie sich, dass Sie den Service 'StockQuoteAxis' in WASCE zur Ausführung des Web-Service-Assistenten implementiert haben.
3. Implementieren Sie den Service, um ihn mit WebSphere MQ Transport for SOAP zu testen, in einem Empfangsprogramm von WebSphere MQ Transport for SOAP für Axis 1.4; siehe „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020.

Informationen zu diesem Vorgang

Die Eclipse IDE für Java EE -Entwickler verwendet Java50 und den Web-Service-Assistenten, um die Proxy-Klassen für den Service zu generieren. 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 mithilfe von **`wsimport`** zu generieren und die erstellten Pakete in Ihre Eclipse Java EE IDE für Webentwickler 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. Es ist kein Anwendungsserver erforderlich. Sie können den Code auch in ein Java-Projekt übertragen und den Buildpfad so konfigurieren, dass er die Axis2 -JAR-Dateien enthält.

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 No auf das Fenster, das Ihnen die Option zum Öffnen der Java EE -Perspektive bietet.

      Die Standardwerte sind so festgelegt, dass WASCE 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. WASCE 2.1 wird 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 **WASCE-Implementierung 1.2** sind bereits überprüft. > **OK** > **Fertigstellen**. Antworten Sie 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 vor der Generierung des Client-Proxy in der Web-Service-Bindung ändern, damit sie auf die URL von WebSphere MQ Transport for SOAP verweist. Der statische Web-Client-Proxy ist dann der Service, der in WebSphere 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. Prüfen Sie, dass die Konfigurationsinformationen zur Web-Service-Seite des Assistenten wie folgt lauten: Server: IBM WASCE v2.1 Server, Web service runtime: Apache Axis2, Client project: StockQuoteAxis2WebClient, Client EAR project: 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 für die Ausführung von WebSphere MQ Transport for SOAP als JMS-Transport.

WebSphere MQ Transport for SOAP stellt zwar einen `transportSender`, jedoch keinen `transportReceiver` zur Verfügung. Das bedeutet also, dass WebSphere MQ Transport for SOAP Axis2-Clients unterstützt. 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 `transportSender` (am Ende der Datei) und suchen Sie nach dem auskommentierten JMS `transportSender` > Klicken Sie mit der rechten Maustaste auf die Zeile > **Hinzufügen vor ...** > **transportSender**.

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 in *MQ_INSTALLATION_PATH*`\java\lib` aus > **OK**.

MQ_INSTALLATION_PATH ist das Verzeichnis, in dem WebSphere MQ installiert ist.

4. Erstellen Sie einen synchronen statischen Client, testen Sie ihn unter Verwendung von HTTP und konvertieren Sie dann den Proxy für die Ausführung des statischen Clients unter Verwendung von WebSphere MQ Transport for SOAP.

a) Klicken Sie mit der rechten Maustaste auf **Java Resources: src** > **Neu** > **Paket** > Paket benennen `soap.client` > Fertig stellen

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**.

Abbildung 182. `SQA2DynamicClient.java`

```
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 in WASCE implementierten StockQuoteAxis-Service und mit WebSphere MQ Transport for SOAP.
   a) Klicken Sie im Projektexplorer mit der rechten Maustaste auf **SQA2StaticClient** > **Ausführen als** > **Java-Anwendung**.

      In der Konsolenansicht wird das Ergebnis `Response is 55.25` angezeigt. Sie können auch das WASCE-Konsolenfenster in der Konsolenansicht auswählen und die Ausgabe auf dem WASCE-Server `StockQuoteAxis called with parameter: ibm`anzeigen.
   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 ändern, damit der Service unter Verwendung von WebSphere 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. Durch die Konfiguration von `axis2.xml` wird Axis2 für die Verwendung von WebSphere MQ Transport for SOAP konfiguriert.
   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 Ausnahmebedingung ausgelöst, da kein konfigurierter `transportSender`-Eintrag für JMS gefunden wird.
      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. Wechseln Sie zur Registerkarte **(x) = Argumente** und geben Sie im Eingabebereich **VM-Argumente** den Pfad zur `axis2.conf` -Datei > **Anwenden** > **Ausführen**ein.
      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 WebSphere MQ Transport for SOAP. Überprüfen Sie diesen, indem Sie sich vergewissern, dass die WASCE-Konsole keine neue Ausgabe enthält. Öffnen Sie das Konsolen-oder Befehlsfenster, das SimpleJavaListener zugeordnet ist, und die Ausgabe dort ist `StockQuoteAxis called with parameter: ibm`.
6. Erstellen Sie einen dynamischen Client für HTTP und WebSphere MQ Transport for SOAP und testen Sie diesen.
   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**.

```
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 den SQA2DynamicClient, WASCE und **SimpleJavaListener**.

7. Erstellen Sie einen asynchronen Client und rufen Sie das Ergebnis in einem Callback-Handler und im Hauptprogrammthread auf.

   Die vom Web-Service-Assistenten für Eclipse Java EE IDE für Webentwickler erstellten asynchronen Client-Proxys unterscheiden sich von den von **wsimport**erstellten Proxys. Die **wsimport** -Proxys verwenden die generischen Typen Future, Responseund AsyncHandler .

   Der Web-Service-Assistent für Eclipse Java EE IDE für Webentwickler erstellt eine abstrakte StockQuoteAxisServiceCallbackHandler -Klasse. 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 183. 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 Beispiel-Proxys werden mit dem Tool **wsimport** generiert, das in 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 184. 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 185. 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 186. 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 187. 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 188. 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 189. 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 unter Verwendung von Eclipse entwickeln
Entwickeln Sie einen Axis 1.4-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln

Entwickeln Sie einen .NET 1- oder 2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

.NET 1- oder 2-Client für WebSphere Transport for SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln

Entwickeln Sie einen .NET 1- oder 2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

Vorbereitende Schritte

Sie können einen .NET 1- oder 2-Client auf verschiedene Arten entwickeln:

1. Generieren Sie Client-Stubs mittels **amqwdeployWMQService** aus einem Web-Service und importieren Sie sie in Visual Studio.
2. Verwenden Sie **java2wsdl** , um WSDL aus einer Java-Implementierung eines Web-Service zu generieren, und verwenden Sie dann `wsdl.exe`, das mit .NET geliefert wird, 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 Visual Studio, um den Client für den Zugriff auf den HTTP-Service zu konfigurieren. Ändern Sie die URL so, dass sie auf den für WebSphere Transport for SOAP implementierten Service verweist.

Die Task verwendet den in „.NET 1- oder 2-Service für den WebSphere MQ-Transport für SOAP mithilfe von Microsoft Visual Studio 2008 entwickeln” auf Seite 1024 entwickelten Service.

Informationen zu diesem Vorgang

Führen Sie die nachfolgenden Schritte aus, um einen .NET 1- oder 2-Client für HTTP und WebSphere MQ Transport for SOAP zu erstellen.

Vorgehensweise

1. Erstellen Sie die Clientkonsolenanwendung und ändern Sie sie so, 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 **Console Application** (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 **Console Application** (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 190 auf Seite 1057.

```
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 190. 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 WebSphere MQ Transport for SOAP abgerufen 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 191. 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 von WebSphere MQ Transport for 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* Das Verzeichnis, in dem WebSphere MQ installiert ist.

3. Testen Sie den Client mit dem Service `StockQuote.asmx` mit WebSphere MQ Transport for 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 2008 die Taste **F5**, um 'StockQuoteClientDotNet' auszuführen.

.NET Framework 1- und .NET Framework 2-Web-Service-Clients

Die mit WebSphere MQ Transport for SOAP bereitgestellten .NET-Beispielclients rufen die Axis- und .NET-Beispielservices mithilfe von generierten Stubs auf.

Für .NET Framework 1- und .NET Framework 2-Clients stellt WebSphere MQ den Zugriff auf Web-Services bereit, die .NET-Clients verwenden. Die Option `genProxiestoDotNet` des Befehls **amqwdeployWMQService** generiert .NET Framework 1- oder .NET Framework 2-Client-Stubs für einen Web-Service. Sie können auch Client-Stubs verwenden, die vom Tool .NET **wsdl** oder von Microsoft Visual Studio 2005 bzw. 2008 generiert wurden.

Die .NET Framework 1- und .NET-Web-Service-Beispielcients sind in *MQ_INSTALLATION_PATH*`\tools\soap\samples\dotnet` installiert.*MQ_INSTALLATION_PATH* Das Verzeichnis, in dem WebSphere MQ installiert ist.

SQVB2Axis.vb
SQVB2Axis.vb,Abbildung 192 auf Seite 1059, ist der Visual Basic-Client zum Aufrufen des Service **StockQuoteAxisService**.

SQVB2DotNet.vb
QVB2DotNet.vb,Abbildung 193 auf Seite 1059, ist der Visual Basic-Client zum Aufrufen des Service **StockQuoteDotNet**.

SQCS2Axis.cs
SQCS2Axis.cs,Abbildung 194 auf Seite 1059, 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 195 auf Seite 1060, 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 192. 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 193. 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 194. 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 195. SQCS2DotNet

Zugehörige Tasks

JAX-RPC-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln
Entwickeln Sie einen Axis 1.4-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP.

JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln
Entwickeln Sie einen Axis2-Web-Service-Client für die Ausführung mit WebSphere MQ Transport for SOAP. Die Axis2-Musterclients, die im Rahmen von WebSphere MQ Transport for SOAP bereitgestellt werden, werden aufgelistet. Außerdem wird der Befehl **wsimport** erläutert, mit dem Proxys generiert werden.

Web-Services mithilfe von WebSphere MQ Transport for SOAP implementieren

Implementieren Sie in einer der unterstützten Serverumgebungen einen Web-Service und stellen Sie über WebSphere 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 WebSphere MQ Transport for SOAP ausgeführt werden soll, kann in verschiedenen SOAP-Laufzeitumgebungen implementiert werden. Unter Axis 1.4 kann ein Service mit der mit WebSphere MQ installierten Software implementiert werden. Für andere Laufzeitumgebungen müssen Sie zusätzliche Software installieren.

Die Ausführung von WebSphere MQ Transport for SOAP ist nicht auf die Server beschränkt, für die Implementierungsanweisungen bereitgestellt werden. 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 WebSphere MQ Transport for SOAP-Bindung. WebSphere MQ bis einschließlich Version 7.0.1.2 unterstützt allerdings nur die Bindung WebSphere MQ Transport for SOAP. Ab Version 7.0.1.3 können Sie Axis2-Clients mit einer URI implementieren, die der Empfehlung des W3C für SOAP over JMS entspricht. Informationen hierzu finden Sie auch im Lernprogramm Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5.

Service mit amqwdeployWMQService für Axis 1.4 für die Verwendung von WebSphere Transport for SOAP implementieren.

Implementieren Sie einen Axis 1.4-Service für WebSphere MQ Transport for SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

Vorbereitende Schritte

1. Folgen Sie den Anweisungen für die Installation von WebSphere MQ Transport for 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 1061 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 1062.

Führen Sie folgenden Anweisungen aus, um einen neuen Axis 1.4-Service für WebSphere MQ Transport for 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 WebSphere 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 install 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 sie 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 WebSphere MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere 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
WebSphere MQ Transport for SOAP ist im WebSphere Application Server in den Service Integration Bus integriert.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP wird von WebSphere ESB und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

.NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren

Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Vorbereitende Schritte

1. Folgen Sie den Anweisungen für die Installation von WebSphere MQ Transport for 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 1063 fort.

Informationen zu diesem Vorgang

Diese Anweisungen beziehen sich auf die erste Implementierung eines .NET-Service. Zum Neustart eines .NET-Service führen Sie das .NET SOAP-Empfangsprogramm erneut aus (Schritt „9” auf Seite 1064).

Führen Sie zur Implementierung eines neuen .NET Framework 1- oder .NET Framework 2-Service für WebSphere 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 *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 196 auf Seite 1064 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 196. Buildbefehl für .NET Framework V2-Service

6. Kopieren Sie die Datei *className*.dll in das Verzeichnis *deployDir**bin*.
7. Richten Sie die WebSphere 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 den generierten .NET-Listener.

```
.\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 WebSphere MQ Transport for 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 for SOAP implementieren
WebSphere 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
WebSphere MQ Transport for SOAP ist im WebSphere Application Server in den Service Integration Bus integriert.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP wird von WebSphere ESB und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

Service für CICS Transaction Server zur Verwendung von WebSphere Transport for SOAP implementieren

WebSphere 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 WebSphere 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 zur Erstellung eines Web-Service den Anweisungen im Abschnitt Erstellen Web-Service erstellen in der CICS-Produktdokumentation.

Informationen zu diesem Vorgang

Folgen Sie den Anweisungen im Abschnitt Configuring CICS zur Verwendung von WebSphere MQ Transport in der CICS-Produktdokumentation. Anhand dieser Anweisungen können Sie den Web-Service für WebSphere 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 WebSphere MQ Transport for SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für WebSphere Application Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP ist im WebSphere Application Server in den Service Integration Bus integriert.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP wird von WebSphere 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

WebSphere MQ Transport for SOAP ist im 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

Führen Sie zur Implementierung eines Service auf dem WebSphere Application Server mit WebSphere MQ Transport for SOAP als SOAP-Transportmethode das folgende Verfahren aus.

Vorgehensweise

1. Konfigurieren Sie WebSphere MQ als JMS-Messaging-Provider für den Service Integration Bus unter WebSphere Application Server.
2. Konfigurieren Sie die für den Service erforderlichen WebSphere MQ-Ressourcen.
3. Folgen Sie den Anweisungen im Abschnitt Configuring JMS-Ressourcen für den synchronen SOAP-over-JMS-Endpunktlister konfigurieren der WebSphere Application Server Network Deployment-Produktdokumentation.

   Für andere WebSphere Application Server-Plattformen stehen entsprechende Anweisungen zur Verfügung.
4. Passen Sie die Service-URI an die WebSphere MQ Transport for SOAP-URI an.
5. Implementieren Sie den Service auf dem 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 WebSphere MQ Transport for SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP ist in die Web-Service-Unterstützung von CICS Transaction Server 4.1 integriert.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren

WebSphere MQ Transport for SOAP wird von WebSphere ESB und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren

Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Vorbereitende Schritte

Für die Aufgabe sind WebSphere Application Server v7.0.0.9 und WebSphere MQ v7.0.1.3 erforderlich.

Informationen zu diesem Vorgang

Die Task besteht aus zwei Schritten:

Vorgehensweise

1. „WebSphere MQ-Ressourcen konfigurieren” auf Seite 1067
2. „WebSphere Application Server-Ressourcen konfigurieren” auf Seite 1068

Nächste Schritte

„WebSphere MQ-Ressourcen konfigurieren” auf Seite 1067

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 WebSphere MQ Transport for SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere 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
WebSphere MQ Transport for SOAP ist im WebSphere Application Server in den Service Integration Bus integriert.

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere MQ Transport for SOAP wird von WebSphere ESB und Process Server nicht direkt unterstützt. Sie müssen hierfür einen benutzerdefinierten Export konfigurieren.

WebSphere MQ-Ressourcen konfigurieren

Vorbereitende Schritte

Zur Unterstützung von Axis2 ist WebSphere MQ 7.0.1.3 oder höher erforderlich.

Informationen zu diesem Vorgang

Der Einfachheit halber setzt die Task voraus, dass WebSphere MQ in derselben Workstation wie die andere Software installiert ist und Bindungsverbindungen verwendet. Die Konfigurationen von WebSphere Application Server und des Axis2-Clients arbeiten mit Clientverbindungen. Um die Task mit Clientverbindungen auszuführen, stellen Sie sicher, dass Sie Nachrichten im Axis-Client und auf WebSphere Application Server-Computern in die Anforderungs- und Antwortwarteschlangen einreihen und aus diesen Warteschlangen abrufen können.

Auch hier wird der Einfachheit halber keine Sicherheitskonfiguration verwendet. Die Benutzer-ID verfügt über die volle `mqm`-Berechtigung.

Vorgehensweise

1. Erstellen Sie den Standardwarteschlangenmanager `QM1`.

   Verwenden Sie WebSphere MQ Explorer, um `QM1` als Standardwarteschlangenmanager zu erstellen. Konfigurieren Sie ihn so, dass er automatisch gestartet wird, und wählen Sie die Option zum Erstellen eines Listeners aus. Alternativ können Sie die folgenden Befehle verwenden:

   ```
   crtmqm -q -sa QM1
   strmqm
   echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
               control(qmgr) replace | runmqsc
   echo start listener(LISTENER.TCP) | runmqsc
   ```

2. Definieren Sie eine Anforderungswarteschlange namens `REQUESTAXIS` und eine Antwortwarteschlange namens `REPLYAXIS`.

   Verwenden Sie den Explorer oder folgende Befehle:

   ```
   echo define ql(REQUESTAXIS) replace | runmqsc
   echo define ql(REPLYAXIS) replace | runmqsc
   ```

Nächste Schritte

„WebSphere Application Server-Ressourcen konfigurieren” auf Seite 1068

WebSphere Application Server-Ressourcen konfigurieren

Vorbereitende Schritte

Für W3C SOAP over JMS-Unterstützung benötigen Sie WebSphere Application Server v7. Diese Konfiguration wurde in WebSphere Application Server Version 7.0 Test Environment v7.0.0.9 Update 1 vorgenommen. WebSphere Application Server wird mit Rational Software Architect for WebSphere Software 7.5.4 bereitgestellt. Durch Anwendung der neuesten verfügbaren Updates wurde Rational Software Architect auf Version 7.5.5.1 aktualisiert.

Erstellen Sie während des Installationsprozesses ein Profil für WebSphere Application Server. In der Task ist keine Verwaltungssicherheit aktiviert. Der Standardprofilname lautet `was70profile1`, der Servername lautet `server1`.

Informationen zu diesem Vorgang

Konfigurieren Sie den WebSphere Application Server. Sie können den Server entweder aus Rational Application Developer und die Verwaltungskonsole aus der Serveransicht starten oder Sie starten den Server mit einer Befehlsdatei und verwalten den Server mithilfe eines Browsers. Die Task verwendet die zweite Methode.

Die Serverbefehlsdateien befinden sich im Ordner *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\bin`. Die Protokolldateien, die Sie überprüfen können, befinden sich im Verzeichnis *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\logs\server1`.

Gemäß Konvention sind alle WebSphere MQ-Objektnamen großgeschrieben und alle JNDI-Namen, die auf die WebSphere MQ-Objekte verweisen, sind kleingeschrieben.

Vorgehensweise

1. Starten Sie den-Server.

   ```
   startServer server1
   ```

2. Starten Sie einen Browser, öffnen Sie die Verwaltungskonsole und melden Sie sich an.

   ```
   http://localhost:9061/ibm/console/unsecureLogon.jsp
   ```

   Geben Sie im Feld 'Benutzer-ID' eine beliebige Zeichenfolge ein.

3. Erstellen Sie die Verbindungsfactory `qm1`
   a) Öffnen Sie im Navigator **Ressourcen** > **JMS** > **Verbindungsfactorys**.
   b) Wählen Sie im Fenster 'Verbindungsfactorys' den Bereich **Node=** ***Knotenname*** aus und klicken Sie auf **Neu**.
   c) Wählen Sie **WebSphere MQ Messaging-Provider** > **OK**aus.
   d) Geben Sie die Verbindungsinformationen für den Warteschlangenmanager über Tabelle 143 auf Seite 1069 > **Weiter**an.

   Tabelle 143. Informationen zur Warteschlangenmanagerverbindung

Name des Felds	**Wert**
Name	`qm1`
JNDI name (JNDI-Name)	`qm1`

   e) Wählen Sie **Alle erforderlichen Informationen in diesen Assistenten eingeben** als Verbindungsaufbaumethode > **Weiter**aus.
   f) Geben Sie `QM1` als Warteschlangenverbindungsdetails ein > **Weiter**.
   g) Geben Sie die Verbindungsdetails unter Tabelle 144 auf Seite 1069 > **Weiter**ein.

   Tabelle 144. Verbindungsangaben

Name des Felds	**Wert**
Übertragung	`Bindings, then client`
Hostname	`localhost`
Port	`1414`
Serververbindungskanal	`SYSTEM.DEF.SVRCONN`

   h) **Testverbindung** > **Weiter** > **Fertigstellen** > **Speichern**.

4. Erstellen Sie die JMS-Anforderungswarteschlange `requestaxis`.
   a) Öffnen Sie im Navigator **Ressourcen** > **JMS** > **Warteschlangen**.
   b) Wählen Sie im Fenster 'Verbindungsfactorys' den Bereich **Node=** ***Knotenname*** aus und klicken Sie auf **Neu**.
   c) Wählen Sie **WebSphere MQ Messaging-Provider** > **OK**aus.
   d) Geben Sie die Warteschlangendetails über Tabelle 145 auf Seite 1069 > **OK** > **Speichern**ein.

   Tabelle 145. Warteschlangendetails

Name des Felds	**Wert**
Name	`requestaxis`

Tabelle 145. Warteschlangendetails (Forts.)	
Name des Felds	**Wert**
JNDI name (JNDI-Name)	`requestaxis`
Warteschlangenname	`REQUESTAXIS`
Name des Warteschlangenmanagers	`QM1`

5. Wiederholen Sie Schritt „4” auf Seite 1069, um die JMS-Antwortwarteschlange `replyaxis` zu erstellen.
6. Erstellen Sie die Aktivierungsspezifikation `qm1as`.

   Die Aktivierungsspezifikation löst die Message-driven Bean (MDB) des Web-Service-Routers aus, wenn eine Nachricht in der Anforderungswarteschlange eintrifft. Die MDB ist im Implementierungsdeskriptor des Web-Service definiert, der vom Web-Service-Assistenten von Rational Application Developer erstellt wurde.

   a) Öffnen Sie im Navigator **Ressourcen** > **JMS** > **Aktivierungsspezifikationen**.

   b) Wählen Sie im Fenster 'Verbindungsfactorys' den Bereich **Node=** ***Knotenname*** aus und klicken Sie auf **Neu**.

   c) Wählen Sie **WebSphere MQ Messaging-Provider** > **OK**aus.

   d) Geben Sie die Basisattribute der Aktivierungsspezifikation über Tabelle 146 auf Seite 1070 > **Weiter**ein.

Tabelle 146. Name der Aktivierungsspezifikation	
Name des Felds	**Wert**
Name	`qm1as`
JNDI name (JNDI-Name)	`qm1as`

   e) Geben Sie die MDB-Informationen über Tabelle 147 auf Seite 1070 > **Weiter**an.

Tabelle 147. MDB-Informationen	
Name des Felds	**Wert**
JNDI-Name des Ziels	`requestaxis`
Nachrichtenselektor	*Bleibt leer*
Bestimmungsorttyp	`Queue`

   f) Wählen Sie **Alle erforderlichen Informationen in diesen Assistenten eingeben** als Verbindungsaufbaumethode > **Weiter**aus.

   g) Geben Sie `QM1` als Warteschlangenverbindungsdetails ein > **Weiter**.

   h) Geben Sie die Verbindungsdetails unter Tabelle 144 auf Seite 1069 > **Weiter**ein.

Tabelle 148. Verbindungsangaben	
Name des Felds	**Wert**
Übertragung	`Bindings, then client`
Hostname	`localhost`
Port	`1414`
Serververbindungskanal	`SYSTEM.DEF.SVRCONN`

   i) **Testverbindung** > **Weiter** > **Fertigstellen** > **Speichern**.

7. Erstellen Sie eine Warteschlangenverbindungsfactory namens `jms/WebServicesReplyQCF` für die Antwortwarteschlange.

   Der Web-Service-Router verwendet eine Warteschlangenverbindungsfactory, um auf eine Antwortwarteschlange zuzugreifen. Im Implementierungsdeskriptor des Web-Services erhält die Warteschlangenverbindungsfactory den JNDI-Standardnamen `jms/WebServicesReplyQCF`. Sie können den Namen im Implementierungsdeskriptor ändern. Fügen Sie in dieser Task den JMS-Ressourcendefinitionen den Standardnamen hinzu.

   a) Öffnen Sie im Navigator **Ressourcen** > **JMS** > **Warteschlangenverbindungsfactorys**.

   b) Wählen Sie im Fenster 'Verbindungsfactorys' den Bereich **Node=** ***Knotenname*** aus und klicken Sie auf **Neu**.

   c) Wählen Sie **WebSphere MQ Messaging-Provider** > **OK**aus.

   d) Geben Sie die Basisattribute der Warteschlangenverbindungsfactory über Tabelle 149 auf Seite 1071 > **Weiter**ein.

   Tabelle 149. Name der Warteschlangenverbindungsfactory

Name des Felds	**Wert**
Name	`WebServicesReplyQCF`
JNDI name (JNDI-Name)	`jms/WebServicesReplyQCF`

   e) Wählen Sie **Alle erforderlichen Informationen in diesen Assistenten eingeben** als Verbindungsaufbaumethode > **Weiter**aus.

   f) Geben Sie `QM1` als Warteschlangenverbindungsdetails ein > **Weiter**.

   g) Geben Sie die Verbindungsdetails unter Tabelle 144 auf Seite 1069 > **Weiter**ein.

   Tabelle 150. Verbindungsangaben

Name des Felds	**Wert**
Übertragung	`Bindings, then client`
Hostname	`localhost`
Port	`1414`
Serververbindungskanal	`SYSTEM.DEF.SVRCONN`

   h) **Testverbindung** > **Weiter** > **Fertigstellen** > **Speichern**.

Nächste Schritte

„JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln” auf Seite 1027

Service für den Serviceendpunkt WebSphere ESB und Process Server zur Verwendung von WebSphere Transport for SOAP implementieren

WebSphere MQ Transport for SOAP wird von WebSphere 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 WebSphere MQ JMS-Export binden können, um einen benutzerdefinierten WebSphere MQ JMS-SOAP-Export zu erstellen.

Folgen Sie zum Erstellen eines benutzerdefinierten Exports für den Empfang von SOAP-Anforderungen über WebSphere MQ Transport for SOAP den folgenden Anweisungen.

Vorgehensweise

1. Lesen Sie die Abschnitte Overview of imports and exports und How to connect to WebSphere MQ in der Produktdokumentation zu WebSphere Process Server for Multiplatforms V6.2.
2. Führen Sie die Task Generieren einer MQ JMS-Exportbindung in der Produktdokumentation zu IBM Business Process ManagerVersion 8.6 aus.

   Verwenden Sie zur Formatierung der SOAP-Nachricht die im Abschnitt Prepackaged JMS data format transformations beschriebene 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 WebSphere MQ Transport for SOAP, indem Sie ein Implementierungsverzeichnis erstellen, den Befehl **amqwdeployWMQService** ausführen und den Axis 1.4-Listener starten.

.NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren
Sie können einen .NET Framework 1- oder 2-Service für WebSphere MQ Transport for SOAP implementieren. Dazu erstellen Sie ein Implementierungsverzeichnis, führen den Befehl **amqwdeployWMQService** aus und starten das .NET-Empfangsprogramm.

Service für CICS Transaction Server zur Verwendung von WebSphere Transport for SOAP implementieren
WebSphere 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
WebSphere MQ Transport for SOAP ist im WebSphere Application Server in den Service Integration Bus integriert.

WebSphere Application Server für die Verwendung von W3C SOAP over JMS konfigurieren
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 1 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Konfigurieren Sie die WebSphere MQ- und WebSphere Application Server-Ressourcen, um den an W3C SOAP over JMS gebundenen Web-Service als Transport zu entwickeln und zu implementieren.

Web-Service-Clients für die Verwendung von WebSphere MQ Transport for SOAP implementieren

Implementieren Sie in einer der unterstützten Clientumgebungen einen Web-Service-Client und stellen Sie über WebSphere MQ Transport for SOAP eine Verbindung zu einem Service her.

Vorbereitende Schritte

Entwickeln Sie den Web-Service und implementieren Sie ihn so, dass er WebSphere MQ Transport for SOAP verwendet.

Informationen zu diesem Vorgang

Ein Web-Service-Client, der mit WebSphere MQ Transport for SOAP ausgeführt werden soll, kann in verschiedenen Clientumgebungen implementiert werden. Sie können einen Java-Client unter Axis 1.4 ausschließlich mit der Software implementieren, die mit WebSphere MQinstalliert wurde. Für andere Clientumgebungen müssen Sie zusätzliche Software installieren.

Die Ausführung von WebSphere Transport for SOAP ist nicht auf die Clientumgebungen beschränkt, für die Implementierungsanweisungen bereitgestellt werden. 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 WebSphere MQ Transport for SOAP-Bindung. WebSphere MQ

bis einschließlich Version 7.0.1.2 unterstützt allerdings nur die Bindung WebSphere MQ Transport for SOAP. Ab Version 7.0.1.3 können Sie Axis2-Clients mit einer URI implementieren, die der Empfehlung des W3C für SOAP over JMS entspricht. Informationen hierzu finden Sie auch im Lernprogramm Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5.

Web-Service-Client für Axis 1.4 zur Verwendung von IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Transport for SOAP an.

Informationen zu diesem Vorgang

Die Bereitstellung eines Axis 1.4-Client für die Verwendung des IBM WebSphere 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 stellt 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 199 auf Seite 1074) eines Axis 1.4 Java-Clients. Der Client (Abbildung 198 auf Seite 1074) ruft einen Web-Service auf, der die Eingabeparameter zurückmeldet. Die Servicedefinition (Abbildung 197 auf Seite 1074) 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 197. 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 198. 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 199. Clientkonfiguration und Ausgabe

Nächste Schritte

1. Wenn Sie den Client als IBM WebSphere 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 von WebSphere MQ Transport for 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS

Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Ändern Sie die URL im für WebSphere 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 WebSphere 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung von WebSphere MQ Transport for 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 WebSphere-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 WebSphere MQ Transport for SOAP verwendet.

Die Task zeigt, wie ein nicht verwalteter Axis2 -Client in Java Standard Editionimplementiert wird. Sie können einen Axis2-Client in einen Web-Container implementieren. In „JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln” auf Seite 1041 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 http://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_container oder in der Dokumentation des Web-Servers.

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 von WebSphere MQ Transport for SOAP genauso wie einen Axis2-Client zur Verwendung von HTTP. Zusätzliche Schritte sind erforderlich, um den WebSphere 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 von WebSphere MQ Transport for 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 203 auf Seite 1077 und Abbildung 204 auf Seite 1078.

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 IBM JREs 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 201 auf Seite 1077 und Abbildung 202 auf Seite 1077 sind Scripts für Windows und Linux.

Vorgehensweise

1. Laden Sie Axis2 1.4.1 von http://ws.apache.org/axis2/download/1_4_1/download.cgi herunter und entpacken Sie das Programm in den 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 WebSphere 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 unter Verwendung von Eclipse entwickeln" auf Seite 1041 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 WebSphere MQ-JAR-Dateien.
6. Starten Sie das Clientprogramm mit dem **Java**-Befehl.

Beispiele

Vier Beispiele zum Ausführen eines Axis2-Clients finden Sie in Abbildung 202 auf Seite 1077 bis Abbildung 204 auf Seite 1078. Abbildung 200 auf Seite 1077 zeigt die Ausgabe der Ausführung des in Abbildung 183 auf Seite 1047 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 200. 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 201. 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 202. 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 203. 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 204. runaxis2.sh: Linux mit Verwendung von 'axis2.sh'

Hinweis

Zugehörige Tasks

Web-Service-Client für Axis 1.4 zur Verwendung von IBM WebSphere 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 WebSphere MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS

Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Ändern Sie die URL im für WebSphere 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 WebSphere 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS

Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Ändern Sie die URL im für WebSphere MQ Transport for SOAP entwickelten Axis2-Client, dass die W3C-Kandidatenempfehlung für SOAP over JMS verwendet wird.

Vorbereitende Schritte

Sie müssen zuerst die Task „JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln” auf Seite 1041 ausführen, um **SimpleJavaListener** mithilfe eines Axis2-Clients und des WebSphere MQ Transport for SOAP-Protokolls aufzurufen.

Zudem müssen Sie den Web-Service erstellt und WebSphere MQ und den WebSphere Application Server in den vorherigen Tasks konfiguriert haben:

1. „WebSphere MQ-Ressourcen konfigurieren” auf Seite 1067.
2. „WebSphere Application Server-Ressourcen konfigurieren” auf Seite 1068.
3. „JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln” auf Seite 1027.

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 vom 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 `jndi-InitialContextFactory` und `jndiURL` hinzufügen, um auf das JNDI-Verzeichnis von WebSphere Application Server 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 WebSphere 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 WebSphere 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 205. 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 unter Verwendung von Eclipse entwickeln” auf Seite 1041ausgeführt haben.
5. Rufen Sie den Service mit der `jndi`-Variante des URI auf, indem Sie WebSphere Application Server als Naming Server verwenden.
   a) Verwenden Sie die URL von `StockQuoteAxis.wsdl`, Abbildung 205 auf Seite 1079 und geben Sie die fehlenden Parameter an, um den Naming Service im WebSphere Application Server zu verwenden.

Die fehlenden Parameter und Werte, die Sie angeben müssen, lauten wie folgt:

Tabelle 151. 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. 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 durch den 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 install directory*\SDP\runtimes\base_v7\runtimes.
- com.ibm.ws.runtime.jar von *Rational install 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 Install 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 unter Verwendung von Eclipse entwickeln” auf Seite 1041ausgeführt 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 WebSphere 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 WebSphere MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung von WebSphere MQ Transport for 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von WebSphere 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client für .NET Framework 1 und 2 zur Verwendung von WebSphere 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 WebSphere-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 WebSphere MQ Transport for SOAP an.

1. Wenn Sie einen Service mit .NET Framework 1 oder 2 implementieren, erstellen Sie den Service als Bibliothek (`.dll`). Implementieren Sie mit WebSphere 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*`\bin`befindet.
4. Setzen Sie die statische Eigenschaft URL im Konstruktor der Client-Proxy-Klasse für WebSphere MQ Transport for SOAP auf die URI `jms:/`.

Informationen zu diesem Vorgang

Für die Implementierung eines Web-Service-Clients für .NET Framework 1 oder 2 zur Verwendung von WebSphere MQ Transport for SOAP ist ein zusätzlicher Implementierungsschritt erforderlich. Sie müssen `amqsoap.dll` bei .NET Framework registrieren. `amqsoap.dll` wird bei der Installation von WebSphere MQ Transport for SOAP automatisch registriert, Sie müssen es jedoch möglicherweise erneut registrieren.

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 208 auf Seite 1082) eines .NET Framework V2-Clients. Der Client (Abbildung 207 auf Seite 1082) ruft einen Web-Service auf, der die Eingabeparameter zurückmeldet. Die statische URL-Definition (Abbildung 206 auf Seite 1082) 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 206. 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 207. 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 208. Konfiguration und Ausgabe

Nächste Schritte

1. Wenn Sie den Client als WebSphere 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 für Axis 1.4 zur Verwendung von IBM WebSphere 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 WebSphere MQ-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Web-Service-Client in Axis2 zur Verwendung von WebSphere MQ Transport for 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 WebSphere-Warteschlangen und -Kanäle, starten Sie den Service und prüfen Sie den Client.

Implementierung in einen Axis2-Client mit W3C SOAP over JMS
Ein an die W3C-Kandidatenempfehlung für SOAP over JMS gebundener Web-Service muss im EJB-Container eines Java EE-Anwendungsservers ausgeführt werden. Diese Task stellt Schritt 4 beim Verbinden eines Axis2-Web-Service-Clients und eines im WebSphere Application Server implementierten Web-Service mit dem W3C SOAP over JMS-Protokoll dar. Ändern Sie die URL im für WebSphere 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 WebSphere 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 v7.5.5.1und WebSphere Application Server Version 7.0 Test Environment v7.0.0.9 Update 1 enthalten ist. Sie benötigen außerdem WebSphere MQ v7.0.1.3.

Die Task baut auf den beiden Tasks „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln" auf Seite 1020 und „JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln" auf Seite 1041 auf. Für die Ausführung dieser Tasks hat Ihre Entwicklungsumgebung bereits Eclipse Galileo, WASCE das Eclipse-Plug-in für WASCE sowie Axis2 1.4.1 installiert. WASCE ist für diese Task nicht erforderlich.

Einige Schritte sind komplex. Es wird vorausgesetzt, dass Sie mit der Entwicklung von Web-Service-Anwendungen für WebSphere Application Server mittels Rational Application Developer vertraut sind. Die Task belegt sehr viel Prozessor- und Speicherplatz. Die Task wurde auf einer virtuellen Maschine mit VMWare Windows XP SP3 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 dem Open-Source-Server WASCE 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 implementieren. Sowohl der Client als auch der Service sollen standardisierte Schnittstellen verwenden. Obwohl Ihr Team für die Entwicklung zukünftiger Projekte eine Lösung implementiert hat, die den WebSphere 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 WebSphere MQ-Transport für SOAP eine Änderung vom HTTP-Client erforderlich machte. Allerdings besteht immer noch das Problem, dass der über den IBM WebSphere MQ-Transport für SOAP verbundene Service von einem speziellen Listener gehostet wird, der von WebSphere MQ bereitgestellt wird: `SimpleJavaListener`.

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 Anwendungsserver 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, gelöst. Wenn Sie eine standardisierte JMS-Nachrichtentransportschnittstelle 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. Die beschriebenen Schritte beziehen sich auf 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 WebSphere MQ-Transport for SOAP eine Verbindung zum Web-Service 'StockQuoteAxis' her.
   a) „JAX-RPC-Service für den WebSphere MQ-Transport für SOAP mithilfe von Eclipse entwickeln” auf Seite 1020
   b) „JAX-WS-Client für WebSphere Transport for SOAP unter Verwendung von Eclipse entwickeln” auf Seite 1041
   c) „Web-Service-Client in Axis2 zur Verwendung von WebSphere MQ Transport for SOAP implementieren” auf Seite 1075
2. Stellen Sie mithilfe eines Axis2-Clients und der W3C-Kandidatenempfehlung für SOAP over JMS eine Verbindung zum Web-Service 'StockQuoteAxis' her.
   a) „WebSphere MQ-Ressourcen konfigurieren” auf Seite 1067
   b) „WebSphere Application Server-Ressourcen konfigurieren” auf Seite 1068
   c) „JAX-WS-EJB-Web-Service für W3C SOAP over JMS entwickeln” auf Seite 1027
   d) „Implementierung in einen Axis2-Client mit W3C SOAP over JMS” auf Seite 1078

WebSphere MQ Bridge for HTTP

Mithilfe von WebSphere MQ Bridge for HTTP können Clientanwendungen Nachrichten mit WebSphere MQ austauschen, ohne dass ein WebSphere MQ-Client installiert werden muss. Sie können WebSphere MQ von jeder Plattform oder in jeder Sprache mit HTTP-Funktionalität aufrufen.

Einführung in WebSphere MQ Bridge for HTTP

WebSphere MQ Bridge for HTTP ist eine JEE-Webanwendung (Java, Enterprise Environment). HTTP-Clients können Anforderungen vom Typ **POST**, **GET** und **DELETE** an die Anwendung senden, um Nachrichten in WebSphere MQ-Warteschlangen einzureihen, zu durchsuchen und zu löschen. WebSphere MQ Bridge for HTTP eignet sich nicht für die Verwendung mit Nachrichten, wenn eine zuverlässige Nachrichtenübermittlung erforderlich ist.

Leistungen

Mit WebSphere MQ Bridge for HTTP können Sie WebSphere MQ-Nachrichten mithilfe von HTTP aus einer großen Vielzahl von Umgebungen senden und empfangen:

- Umgebungen, die HTTP, nicht jedoch WebSphere MQ unterstützen.
- Umgebungen, die nicht genügend Speicherplatz zum Installieren eines WebSphere MQ-Clients aufweisen.
- Zu viele Umgebungen, um den WebSphere MQ-Client auf allen Systemen zu installieren, die Zugriff auf WebSphere MQ benötigen.
- Webbasierte Anwendungen, über die Sie Nachrichten ohne Codierung der eigenen Bridge zu WebSphere MQ senden oder empfangen möchten.
- Webbasierte Anwendungen, die Sie mit asynchronen Verfahren wie AJAX erweitern möchten. WebSphere MQ Bridge for HTTP stellt WebSphere MQ-Warteschlangen und -Themen mithilfe von Representation State Transfer (REST) über HTTP zur Verfügung.

HTTP-Support kann sowohl bei Punkt-zu-Punkt- als auch bei Publish/Subscribe-Messaging-Topologien genutzt werden.

Wie funktioniert HTTP-Support?

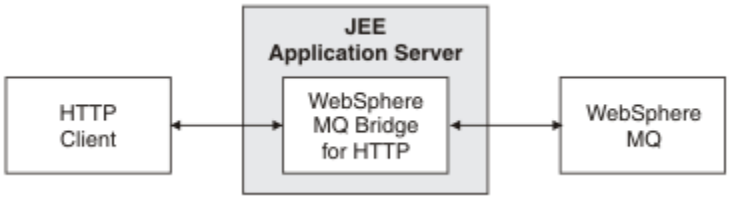


Abbildung 209. WebSphere MQ Bridge for HTTP

Die Webanwendung WebSphere MQ Bridge for HTTP empfängt HTTP-Anforderungen von einem oder mehreren Clients. Sie kommuniziert mit WebSphere MQ im Auftrag der Clients und gibt den Clients HTTP-Antworten zurück.

Bei WebSphere MQ Bridge for HTTP handelt es sich um ein JEE-Servlet, das über einen Ressourcenadapter mit WebSphere MQ verbunden ist. Das HTTP-Servlet verarbeitet die HTTP-Anforderungstypen **POST**, **GET** und **DELETE**.

Tabelle 152. WebSphere 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.
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 eine HTTP- **POST**-Anforderung mithilfe eines Browserformulars oder eines AJAX-Toolkits erstellen.

In Abbildung 210 auf Seite 1085 ist eine HTTP-Anforderung dargestellt, mit der eine Nachricht in die Warteschlange `myQueue` gestellt wird. Diese Anforderung enthält den HTTP-Header `x-msg-correlId` zum Festlegen der Korrelations-ID der WebSphere MQ-Nachricht.

```
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 210. Beispiel für eine HTTP-Anforderung vom Typ **POST** an eine Warteschlange*

Abbildung 211 auf Seite 1086 zeigt die Antwort an, die dem Client zurückgegeben wird. 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 211. 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 eine HTTP- **DELETE**-Anforderung mithilfe eines Browserformulars oder eines AJAX-Toolkits erstellen.

In Abbildung 212 auf Seite 1086 ist eine HTTP-Anforderung zum Löschen der nächsten Nachricht in der Warteschlange `myQueue` dargestellt. Als Antwort wird der Nachrichtentext an den Client zurückgegeben. Aus Sicht von WebSphere MQ ist HTTP **DELETE** ein Abruf mit Löschen.

Die Anforderung enthält den HTTP-Anforderungsheader `x-msg-wait`, der WebSphere MQ Bridge for HTTP mitteilt, 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 212. Beispiel für eine HTTP-Anforderung vom Typ **DELETE***

Abbildung 213 auf Seite 1086 ist die Antwort, die an den Client zurückgegeben wird. 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 213. 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. Aus Sicht von WebSphere MQ ist HTTP **GET** eine Anforderung zum Durchsuchen. Sie können eine HTTP-Anforderung vom Typ **GET** mithilfe eines Java-Clients, eines Browserformulars oder eines AJAX-Toolkits erstellen.

In Abbildung 214 auf Seite 1087 ist eine HTTP-Anforderung zum Durchsuchen der nächsten Nachricht in der Warteschlange `myQueue` dargestellt.

Die Anforderung enthält den HTTP-Anforderungsheader `x-msg-wait`, der WebSphere MQ Bridge for HTTP mitteilt, 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 214. Beispiel für eine HTTP-Anforderung vom Typ **GET***

Abbildung 215 auf Seite 1087 ist die Antwort, die an den Client zurückgegeben wird. 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 215. Beispiel für eine HTTP-Antwort vom Typ **GET***

WebSphere MQ Bridge for HTTP installieren, konfigurieren und überprüfen

Rufen Sie WebSphere MQ Bridge for HTTP ab, indem Sie "Java Messaging and Web Services" entweder aus dem WebSphere MQ MQI-Client-oder Serverinstallationsmaterial installieren. Implementieren Sie WebSphere MQ Bridge for HTTP auf einem geeigneten Anwendungsserver.

Vorbereitende Schritte

Überprüfen Sie die vorausgesetzten Produkte im Abschnitt Systemvoraussetzungen für IBM WebSphere MQ. Der Installationsprozess prüft nicht, ob die zum Ausführen von WebSphere MQ Bridge for HTTP erforderlichen Softwareprodukte vorhanden und verfügbar sind. Sie müssen überprüfen, ob die Voraussetzungen installiert sind.

WebSphere MQ Bridge for HTTP kann auf jedem Java EE 1.4-kompatiblen Anwendungsserver durch Installation des WebSphere MQ-Ressourcenadapters ausgeführt werden. Sie können WebSphere MQ Bridge for HTTP auch unter einer älteren WebSphere Application Server-Version als Version 6.0.2.1 ausführen. Verwenden Sie den Nachrichten-Listener-Port von WebSphere Application Server, um WebSphere MQ als JMS-Provider zu integrieren.

WebSphere MQ Bridge for HTTP wird nur für folgende Anwendungsserver unterstützt:

- WebSphere Application Server 6.0.2.1 und höher.
- WebSphere Application Server Community Edition Version 1.1 und höher.

Informationen zu diesem Vorgang

WebSphere MQ Bridge for HTTP wird als `.war`-Datei (`WMQHTTP.war`) bereitgestellt.

- Auf UNIX-Plattformen und Linux
  - `WMQHTTP.war` ist Teil der Installationsoption "Java Messaging and Web Services" . Diese Option ist im Installationsprogramm des Clients und des Servers enthalten.
  - `WMQHTTP.war` wird im Verzeichnis *<mqmtop>*`/java/http/WMQHTTP.war`installiert. *< mqmtop>* ist das Verzeichnis, in dem WebSphere MQ installiert ist.
  - `WMQHTTP.samples` wird im Verzeichnis *<mqmtop>*`/java/http/samples`installiert. *< mqmtop>* ist das Verzeichnis, in dem WebSphere MQ installiert ist.

Führen Sie die folgenden Installationsschritte durch, um WebSphere MQ Bridge for HTTP zu installieren, implementieren und konfigurieren und um die Konfiguration zu prüfen. Die einzelnen Schritte der Konfiguration sind je nach Anwendungsserver unterschiedlich. Verwenden Sie „WebSphere MQ Bridge for HTTP unter WebSphere Application Server V6.1.0.9 implementieren und überprüfen" auf Seite 1088 als Vorlage für die auf Ihrem Anwendungsserver durchzuführenden Schritte.

Vorgehensweise

1. Sie erhalten `WMQHTTP.war`, indem Sie den WebSphere MQ-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 ggf. WebSphere MQ als Ressourcenadapter auf Ihrem Anwendungsserver.

   Ermitteln Sie, ob WebSphere MQ auf Ihrem Anwendungsserver bereits als Messaging-Provider konfiguriert ist. Suchen Sie mithilfe des mit Ihrem Anwendungsserver ausgelieferten Verwaltungstools nach WebSphere MQ. WebSphere MQ befindet sich möglicherweise unter dem Pfad **Ressourcen > JMS > Messaging-Provider**.
5. Konfigurieren Sie eine Verbindungsfactory auf dem Anwendungsserver, um eine Verbindung zu einem Warteschlangenmanager herzustellen, der den Clienttransport WebSphere MQ verwendet.[12].
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: „WebSphere MQ Bridge for HTTP unter WebSphere Application Server V6.1.0.9 implementieren und überprüfen" auf Seite 1088.
2. Führen Sie die HTTP-Java-Beispielanwendungen aus.

WebSphere MQ Bridge for HTTP unter WebSphere Application Server V6.1.0.9 implementieren und überprüfen

Verwenden Sie das folgende Beispiel, um eine Implementierung von WebSphere MQ Bridge for HTTP für die Ausführung der HTTP-Java-Beispielprogramme vorzubereiten. Die Implementierung befindet sich auf WebSphere Application Server V6.1.0.9.

[12] Konfigurieren Sie zunächst mindestens den Clienttransport. Einige Anwendungsserver können eine Verbindung zu WebSphere MQ über direkte Verbindungen oder Verbindungen im Bindungsmodus herstellen.

Vorbereitende Schritte

1. Folgen Sie den Anweisungen im Abschnitt „WebSphere MQ Bridge for HTTP installieren, konfigurieren und überprüfen” auf Seite 1087, 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 153 auf Seite 1089 konfiguriert:

Tabelle 153. WS-Manager-Konfiguration	
Objekt	**Wert**
Hostname	`itso-01`
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 WebSphere 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

Im Beispiel werden die hierfür erforderlichen Schritte gezeigt, wenn WebSphere Application Server V6.1.0.9 als Anwendungsserver verwendet wird. Wenn Sie eine andere WebSphere Application Server-Version oder einen anderen Anwendungsserver verwenden, weicht das Verfahren hiervon ab. WebSphere Application Server V6.1.0.9 wird mit WebSphere MQ (installiert als Messaging-Provider) vorkonfiguriert, wobei die WebSphere MQ-Client--Bibliotheken verwendet werden. Falls WebSphere MQ nicht als Messaging-Provider vorkonfiguriert ist oder wenn Sie WebSphere MQ-Serverbindungen verwenden möchten, müssen Sie auf Ihrem Anwendungsserver den WebSphere MQ-Ressourcenadapter für JEE installieren.

Folgen Sie den Anweisungen zur Implementierung von WebSphere MQ Bridge for HTTP in WebSphere Application Server V6.1.0.9 und prüfen Sie die Implementierung in einem Browser:

Vorgehensweise

1. Klicken Sie im Navigationsfenster auf **Ressourcen > JMS-Provider > WebSphere MQ Messaging-Provider**.

   Je nach WebSphere Application Server-Implementierung 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 für JMS-Provider die Informationen in Tabelle 154 auf Seite 1090oder Alternativen Ihrer Wahl an und klicken Sie auf **Anwenden > Speichern**.

Tabelle 154. 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 Teil der URI und gibt WebSphere MQ Bridge for HTTP an. 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 **Servern Module zuordnen** einen Cluster oder Server aus, aktivieren Sie das Auswahlfeld und klicken Sie auf **Anwenden > Weiter**.
8. Klicken Sie auf der Seite **Ressourcenreferenzen zu Ressourcen zuordnen** im Formular **javax.jms.ConnectionFactory** auf **Durchsuchen ...** in der Zeile IBM WebSphere 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 **Auswählen** für IBM WebSphere MQ Bridge for HTTP und klicken Sie auf **Weiter > Weiter > Fertigstellen > 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 WebSphere MQ Bridge for HTTP

WebSphere MQ Bridge for HTTP verwendet die WebSphere MQ-Klassen für die JMS-Publish/Subscribe-Schnittstelle. Mit HTTP **POST** wird eine Veröffentlichung erstellt. Mit HTTP **DELETE** wird eine nicht perma-

nente Subskription erstellt. Sie müssen Publish/Subscribe für JMS konfigurieren, um die Themen-URI verwenden zu können.

Publish/Subscribe ist in Version 7 vollständig in WebSphere MQ integriert. Vor Version 7 bearbeitete ein separater Publish/Subscribe-Broker Veröffentlichungen und Subskriptionen. Es wird als "eingereihtes" Publish/Subscribe bezeichnet, um es von dem vollständig integrierten Publish/Subscribe in Version 7 zu unterscheiden. Version 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 Version 6 wurde der Broker mit WebSphere MQ beliefert; vor Version 6 war er als Support-Pack verfügbar.

Konfiguration

WebSphere MQ Bridge for HTTP verwendet die JMS-Schnittstelle, um zu publizieren und zu subskribieren. In Version 7 können Sie steuern, ob die WebSphere MQ Klassen für JMS das eingereihte oder integrierte Publish/Subscribe mit der JMS-Eigenschaft `PROVIDERVERSION` verwenden.

Sie können außerdem entscheiden, ob Sie die WebSphere MQ-Client-Bibliotheken mit WebSphere MQ Bridge for HTTP oder Serverbibliotheken verwenden möchten. Clientbibliotheken der Version 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 WebSphere MQ als Messaging-Provider verwenden, nutzen dabei Clientbibliotheken. Damit integriertes Publish/Subscribe verwendet werden kann, müssen die MQI-Client-und -Serverbibliotheken von WebSphere MQ 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 155 auf Seite 1091). Überprüfen Sie, welche Bibliotheken mit dem von Ihnen verwendeten Web-Server oder Anwendungsserver installiert oder konfiguriert sind.

Tabelle 155. 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
```

WebSphere 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.

Beispiele für die WebSphere MQ Bridge for HTTP ausführen

Die Beispiele für die WebSphere MQ Bridge for HTTP sind nur für die Verwendung auf dem Betriebssystem Windows verfügbar. Die Beispiele zeigen, wie HTTP- **POST** und HTTP- **DELETE** Befehle aus Java-Programmen an WebSphere MQ Bridge for HTTP übergeben werden.

Vorbereitende Schritte

Prüfen Sie Ihre Installation der WebSphere MQ Bridge for HTTP und führen Sie dazu den Schritt „7” auf Seite 1088 im Abschnitt „WebSphere MQ Bridge for HTTP installieren, konfigurieren und überprüfen” auf Seite 1087 aus.

Die HTTP-Beispiele werden in den Verzeichnissen installiert, die in Tabelle 156 auf Seite 1092 gezeigt werden. In jedem Fall wird der Quellcode im Unterverzeichnis `/src` installiert.

Tabelle 156. Position der HTTP-Beispiele

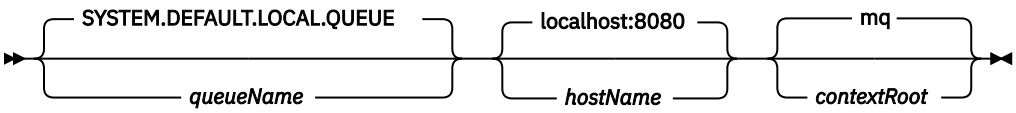
Plattform	**Position**
Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
Alle anderen Plattformen	*MQ_INSTALLATION_PATH*/samp/http
MQ_INSTALLATION_PATH ist das Verzeichnis, in dem WebSphere MQ installiert ist.	

Informationen zu diesem Vorgang

Die Beispiele simulieren die WebSphere MQ-Beispielanwendungen AMQSPUT und AMQSGET. Sie veranschaulichen die folgenden Funktionen in einer Punkt-zu-Punkt-Messaging-Umgebung:

- **HTTPPOST** -Sendet HTTP **POST** -Anforderungen in einer Java-Anwendung zum Einreihen von Nachrichten in eine WebSphere MQ -Warteschlange unter Verwendung der WebSphere MQ -Bridge für HTTP und verarbeitet die Antworten.
- **HTTPDELETE** -Sendet HTTP-Anforderungen **DELETE** in einer Java-Anwendung zum Abrufen von Nachrichten aus einer WebSphere MQ -Warteschlange unter Verwendung der WebSphere MQ -Bridge für HTTP und verarbeitet die Antworten, die die WebSphere 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 WebSphere MQ-Warteschlange zu übergeben.
   a) Wenn Sie eine weitere Nachricht senden möchten, geben Sie weiteren Text ein.
      Der Text bildet den Hauptteil einer zweiten WebSphere MQ-Nachricht.
   b) Drücken Sie die Eingabetaste, um die Nachricht an die WebSphere 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

Bei dem **HTTPDELETE**-Beispiel wird ein Abruf mit Löschen aller Nachrichten ausgeführt, die Sie in die WebSphere MQ-Warteschlange gestellt 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 WebSphere 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 für die WebSphere Bridge for HTTP

Die standardmäßigen Aspekte zur Websicherheit gelten für die Authentifizierung eines Web-Browserclients. Die Autorisierung für WebSphere MQ-Ressourcen besteht auf der Ebene des Benutzers, der das WebSphere Bridge for HTTP-Servlet ausführt, und nicht auf der Ebene des einzelnen Web-Browserclients. Für WebSpher MQ gelten standardmäßige Sicherheitsaspekte.

Die Datenübertragung von einem Web-Browser an eine WebSphere MQ-Anwendung und umgekehrt unter Verwendung der WebSphere Bridge for HTTP erfolgt in drei Schritten:

Clientverbindung
Vom Browser zur WebSphere Bridge for HTTP über eine TCP/IP-Verbindung unter Verwendung von HTTP.

Ressourcenadapterverbindung zu WebSphere MQ
Die Verbindung erfolgt von der WebSphere Bridge for HTTP zu einem WebSphere MQ-Warteschlangenmanager. Bei der Verbindung handelt es sich um eine Clientverbindung, eine Verbindung über TCP/IP oder eine lokale WebSphere MQ-Bindungsverbindung. Sobald die Verbindung hergestellt

wurde, wird die HTTP-Anforderung in eine standardmäßige lokale Warteschlange oder eine Übertragungswarteschlange eingereiht.

Von der lokalen WebSphere MQ-Warteschlange über einen oder mehrere Kanäle zur 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 WebSphere 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 WebSphere Bridge for HTTP zu identifizieren. Die Benutzer-ID muss über eingeschränkte WebSphere MQ-Berechtigungen nur für 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. Die WebSphere Bridge for HTTP stellt einen Angriffspunkt zwischen dem Internet und Ihrem Intranet dar.

Die Vorgehensweise zum Sichern der Verbindung zwischen Ihrem Ressourcenadapter und WebSphere MQ ist von Ihrem speziellen Ressourcenadapter abhängig. Weitere Informationen finden Sie in der Dokumentation zu für den Ressourcenadapter.

Component Object Model-Schnittstelle verwenden (WebSphere MQ Automation Classes for ActiveX)

WebSphere MQ Automation Classes for ActiveX (MQAX) sind ActiveX-Komponenten, die Klassen bereitstellen, die Sie in Ihrer Anwendung für den Zugriff auf WebSphere MQ verwenden können.

Für MQAX sind eine WebSphere MQ-Umgebung und eine entsprechende WebSphere MQ-Anwendung erforderlich, mit der die Kommunikation erfolgen soll.

Ihre ActiveX-Anwendung erhält so die Möglichkeit, auf allen Ihren Unternehmenssystemen, auf die Sie über WebSphere MQ zugreifen können, Transaktionen auszuführen und Daten abzurufen.

WebSphere MQ Automation Classes for ActiveX:

- Ermöglichen den Zugriff auf die Funktionen und Features der WebSphere MQ-API und damit vollständige Verbindungsflexibilität mit anderen WebSphere MQ-Plattformen.
- Entspricht den normalen Konvention, die von einer ActiveX-Komponente erwartet werden.
- Entspricht dem Objektmodell WebSphere MQ , das auch für .NET, C++, Java und LotusScriptverfügbar ist.

Beispiele für den MQAX-Starter werden bereitgestellt. Mithilfe dieser Beispiele können Sie zunächst überprüfen, ob Ihre Installation von MQAX erfolgreich war und die grundlegende WebSphere MQ-Umgebung eingerichtet ist. Die Beispiele veranschaulichen außerdem, wie MQAX verwendet werden kann.

Scripterstellung für COM und ActiveX

Das Component Object Model (COM) ist ein von Microsoft definiertes objektbasiertes Programmiermodell. 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.

ActiveX ist eine Gruppe von auf COM basierenden Technologien, die Anwendungsentwicklung, wiederverwendbare Komponenten und Internet-Technologien auf den Microsoft Windows-Plattformen integrieren. 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 WebSphere MQ-Umgebung

WebSphere MQ Automation Classes for ActiveX können nur mit **32-Bit**-ActiveX-Scripting-Clients verwendet werden.

Die COM-Komponente kann nur für **32-Bit**-Anwendungen verwendet werden. Wenn Sie eine 64-Bit-COM-Anwendung schreiben möchten, können Sie die .NET-Schnittstelle verwenden.

Zur Ausführung von MQAX in einer WebSphere MQ-Serverumgebung muss auf Ihrem System Windows 2000 oder eine höhere Version installiert sein.

Zur Ausführung von MQAX in einer WebSphere MQ-MQI-Clientumgebung muss der WebSphere MQ-MQI-Client auf Windows 2000 oder einer höheren Version auf Ihrem System installiert sein:

Der WebSphere MQ-MQI-Client erfordert Zugriff auf mindestens einen WebSphere MQ-Server. Wenn Sie sowohl den WebSphere MQ-MQI-Client als auch den WebSphere MQ-Server auf Ihrem System installiert haben, werden MQAX-Anwendungen immer auf dem Server ausgeführt. Die ActiveX-Schnittstelle zu MQAI ist nur in WebSphere MQ-Serverumgebungen verfügbar.

Design und Programmierung mithilfe von WebSphere MQ Automation Classes for ActiveX

MQAX-Anwendungen entwerfen, die auf Nicht-ActiveX-Anwendungen zugreifen

Über die WebSphere MQ Automation Classes ist ein Zugriff auf die Funktionen der WebSphere MQ-API möglich. Somit können Sie alle Vorteile der Verwendung von WebSphere MQ für Ihre Windows-Anwendung nutzen.

Die Gesamtarchitektur Ihrer Anwendung ist dieselbe wie bei jeder WebSphere MQ-Anwendung, Sie sollten also alle im Abschnitt „Anwendungen entwickeln” auf Seite 7 beschriebenen Gestaltungsaspekte berücksichtigen.

Zur Verwendung der WebSphere MQ Automation Classes codieren Sie die Windows-Programme in Ihrer Anwendung in einer Sprache, die die Erstellung und Verwendung von COM-Objekten unterstützt. Beispiel: Visual Basic, Java und andere ActiveX Scripting-Clients. Die Klassen können anschließend ohne großen Aufwand in Ihre Anwendung integriert werden, da die benötigten WebSphere MQ-Objekte mithilfe der nativen Syntax der Implementierungssprache codiert werden können.

WebSphere MQ Automation Classes for ActiveX verwenden

Beim Entwerfen einer ActiveX-Anwendung, die WebSphere MQ Automation Classes for ActiveX verwendet, stellt die Nachricht, die vom fernen WebSphere MQ-System gesendet oder empfangen wird, die wichtigste Information dar. Deshalb müssen Sie das Format der Elemente kennen, die in die Nachricht integriert werden. Damit ein MQAX-Script funktioniert, müssen sowohl das Script als auch die WebSphere 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.

Wenn es darum geht, wie die Implementierung des von Ihnen gestalteten Systems strukturiert werden soll, sollten Sie berücksichtigen, dass Ihre MQAX-Scripts auf dem System ausgeführt werden, auf dem auch der WebSphere MQ-Warteschlangenmanager oder der WebSphere MQ-Client installiert ist.

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 WebSphere MQ-Anwendung erstellt wurde und diese Eigenschaften von WebSphere MQ generiert werden, ist es besser, die Eigenschaften AccountingTokenHex, CorrelationId-Hex, GroupIdHex und MessageIdHex zu verwenden, um die zugehörigen Werte anzuzeigen oder in irgend-einer Form zu bearbeiten, einschließlich Ihrer Rückgabe in einer Nachricht an WebSphere MQ. Dies liegt daran, dass es sich bei den von WebSphere MQ generierten Werten um Bytezeichenfolgen mit Werten von 0 bis einschließlich 255 und nicht um Zeichenfolgen druckbarer Zeichen handelt.

Wenn eine Nachricht von Ihrem MQAX-Script erstellt wurde, können Sie die Eigenschaften AccountingTo-ken, CorrelationId, GroupId und MessageId oder deren hexadezimale Entsprechung verwenden.

WebSphere MQ-Konstanten

WebSphere MQ-Konstanten werden als Elemente der WebSphere MQ-Aufzählung in Bibliothek MQAX200 bereitgestellt.

WebSphere MQ-Zeichenfolgekonstanten

WebSphere MQ-Zeichenfolgekonstanten und die entsprechenden Zeichenfolgen.

WebSphere MQ-Zeichenfolgekonstanten sind bei Verwendung von WebSphere 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:

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_STRIsNG	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

Konstanten für Nullzeichenfolgen

Die WebSphere 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 WebSphere MQ Automation Classes for ActiveX nicht unterstützt. Das Festlegen der Konstanten auf eine leere Zeichenfolge hat die gleiche Auswirkung.

Um beispielsweise die verschiedenen IDs einer MQMessage auf diese Werte zu setzen: *mymessage*.**MessageId** = "" *mymessage*.**CorrelationId** = "" *mymessage*.**AccountingToken** = ""

Nachricht von WebSphere MQ empfangen

Es gibt verschiedene Möglichkeiten, eine Nachricht von WebSphere 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 Ihre anderen Threads allerdings Zugriff auf WebSphere MQ benötigen, ist eine zweite Verbindung zu WebSphere MQ über weitere MQAX-Warteschlangenmanager und -Warteschlangenobjekte erforderlich.

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 WebSphere 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 mithilfe der bereitgestellten WebSphere MQ-Konstanten festgelegt und interpretiert werden.Abbildung 216 auf Seite 1098 zeigt ein Beispiel für Folgendes:

```
/* 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 216. Bereitgestellte WebSphere MQ-Konstanten für die Codierung

So können Sie beispielsweise eine ganze Zahl von einem Intel-System in System/390-Codierung an ein System/390-Betriebssystem senden:

```
Dim msg As New MQMessage   'Define a WebSphere 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 WebSphere MQ Automation Classes for ActiveX erfolgt die Zeichensatzkonvertierung mithilfe von Konvertierungstabellen.

So werden beispielsweise Zeichenfolgen automatisch in Codepage 437 konvertiert:

```
Dim msg As New MQMessage                 'Define a WebSphere 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" in dem 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.

Ähnlich gilt: Bei Verwendung der ReadString-Methode enthält die eingehende Nachricht einen Zeichensatz, der aus dem Wert des WebSphere MQ-Nachrichtendeskriptors (MQMD) erstellt wurde, und es erfolgt eine Konvertierung von dieser Codepage in Unicode, bevor die Nachricht in Ihre Scripting-Sprache zurückgegeben wird.

Threading

Mit WebSphere MQ Automation Classes for ActiveX wird ein Free-Threading-Modell implementiert, bei dem Objekte zwischen Threads verwendet werden können.

Zwar ist die Verwendung von MQQueue- und MQQueueManager-Objekten in MQAX möglich, WebSphere MQ lässt derzeit jedoch die gemeinsame Nutzung von Handles zwischen verschiedenen Threads nicht zu.

Wenn versucht wird, die Handles in einem anderen Thread zu verwenden, kommt es zu einem Fehler und WebSphere MQ meldet den Rückgabecode MQRC_HCONN_ERROR.

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 1099.

   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 WebSphere 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 WebSphere 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.

WebSphere MQ Automation Classes for ActiveX - Referenz

In diesem Abschnitt werden die Klassen der WebSphere MQ Automation Classes for ActiveX (MQAX) beschrieben, die für ActiveX entwickelt wurden. Mit den Klassen können Sie ActiveX-Anwendungen schreiben, die über WebSphere MQ auf andere Anwendungen zugreifen können, die in Ihren Nicht-ActiveX-Umgebungen ausgeführt werden.

Die WebSphere MQ Automation Classes for ActiveX-Schnittstelle

In WebSphere MQ Automation Classes for ActiveX werden vordefinierte numerische ActiveX-Konstanten (wie z. B. MQMT_REQUEST) bereitgestellt, die zur Verwendung der Klassen benötigt werden.

Die ActiveX-Automatisierungsklassen setzen sich aus den folgenden Klassen zusammen:

- „MQSession-Klasse” auf Seite 1102
- „MQQueueManager-Klasse” auf Seite 1106
- „MQQueue-Klasse” auf Seite 1117
- „Klasse MQMessage” auf Seite 1132
- „MQPutMessageOptions-Klasse” auf Seite 1154
- „Klasse 'MQGetMessageOptions'” auf Seite 1156
- „Klasse MQDistributionList” auf Seite 1158
- „Klasse MQDistributionListItem” auf Seite 1162

Darüber hinaus werden in WebSphere MQ Automation Classes for ActiveX vordefinierte numerische ActiveX-Konstanten (wie z. B. MQMT_REQUEST) bereitgestellt, die zur Verwendung der Klassen erforderlich sind. Diese werden in der MQ-Aufzählung in der Bibliothek MQAX200 bereitgestellt. Bei den Konstanten handelt es sich um eine Untergruppe der in den C-Headerdateien von WebSphere MQ (cmqc*.h) definierten Konstanten mit einigen zusätzlichen Ursachencodes für WebSphere MQ Automation Classes for ActiveX.

Informationen zu den Klassen von WebSphere MQ Automation Classes for ActiveX

Lesen Sie diese Informationen zusätzlich zu den Referenzabschnitten unter Anwendungsreferenz entwickeln.

Wichtige Informationen finden Sie unter Features, die nur mit der primären Installation unter Windows verwendet werden können .

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 1099.

Über die MQQueueManager- und MQQueue-Klassen ist ein Zugriff auf die zugrunde liegenden WebSphere MQ-Objekte möglich. Bei Methoden- oder Eigenschaftszugriffen für diese Klassen kommt es im Allgemeinen zu Aufrufen über das WebSphere MQ-MQI.

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 WebSphere MQ-Aufruf. Jede dieser Methoden erstellt ein Objekt mit Referenzinformationen, gefolgt von einer Verbindungsherstellung oder dem Öffnen eines WebSphere MQ-Objekts.

Whenn eine Verbindung zu einem Warteschlangenmanager hergestellt wird, enthält sie das von WebSphere MQ generierte Verbindungskennungsattribut ('connection handle').

Wenn eine Warteschlange geöffnet wird, enthält sie das von WebSphere MQ generierte Objektkennungsattribut ('object handle').

Diese WebSphere MQ-Attribute stehen für das 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 WebSphere MQ ActiveX-Klassen enthalten zwei spezielle schreibgeschützte Eigenschaften - ReasonCode und CompletionCode. Diese Eigenschaften können jederzeit gelesen werden.

Beim Versuch, auf eine andere Eigenschaft zuzugreifen oder einen Methodenaufruf auszugeben, könnte ein WebSphere MQ-Fehler generiert werden.

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 WebSphere 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 1103.
- „Eigenschaft ExceptionThreshold” auf Seite 1103.
- „Eigenschaft 'ReasonCode'” auf Seite 1104.
- „Eigenschaft ReasonName” auf Seite 1104.

Methode

- „AccessGetMessageOptions-Methode” auf Seite 1104.
- „AccessMessage-Methode” auf Seite 1105.
- „AccessPutMessageOptions-Methode” auf Seite 1105.
- „AccessQueueManager-Methode” auf Seite 1105.
- „ClearErrorCodes-Methode” auf Seite 1105.
- „ReasonCodeName-Methode” auf Seite 1105.

CompletionCode, Eigenschaft

Schreibgeschützt. Gibt den WebSphere MQ-Beendigungscode zurück, der von der aktuellsten Methode oder Eigenschaftengruppe festgelegt wurde, die für ein WebSphere 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 1099.

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 WebSphere 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**

Zum Festlegen: *MQSession*.**ExceptionThreshold** = *ExceptionThreshold$*

Eigenschaft 'ReasonCode'

Schreibgeschützt. Gibt den Ursachencode zurück, der von der aktuellsten Methode oder Eigenschaftengruppe festgelegt wurde, die für ein WebSphere 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 1099.

Definiert in: MQSession-Klasse

Datentyp: Lang

Werte:

- Informationen hierzu finden Sie unter Ursache (MQLONG) und in den zusätzlichen MQAX-Werten, die unter „Ursachencodes" auf Seite 1170 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 1099.

Definiert in: MQSession-Klasse

Datentyp: String

Werte:

- Siehe API-Ursachencodes.

Syntax: Für den Abruf: *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 verbindet es über den WebSphere MQ-MQI-Client oder den WebSphere MQ-Server mit einem echten Warteschlangenmanager. Neben der Verbindungsherstellung führt diese Methode auch das Öffnen des Warteschlangenmanagerobjekts aus.

Wenn sowohl der WebSphere MQ-MQI-Client als auch der WebSphere 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 WebSphere MQ, amqzst.dll zu laden. Wenn diese DLL nicht vorhanden ist (z. B. bei einer reinen Clientinstallation), versucht WebSphere MQ, 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 (WebSphere MQ-Server) oder remote ausgeführt werden, wobei der Zugriff über den WebSphere 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 1108.
- „AuthorityEvent, Eigenschaft” auf Seite 1108.
- „Eigenschaft BeginOptions” auf Seite 1108.
- „Eigenschaft ChannelAutoDefinition” auf Seite 1108.
- „Eigenschaft ChannelAutoDefinitionEvent” auf Seite 1109.
- „ChannelAutoDefinitionExit, Eigenschaft” auf Seite 1109.
- „Eigenschaft 'CharacterSet'” auf Seite 1109.
- „Eigenschaft 'CloseOptions'” auf Seite 1109.
- „CommandInputQueueName, Eigenschaft” auf Seite 1109.
- „Eigenschaft CommandLevel” auf Seite 1109.
- „CompletionCode, Eigenschaft” auf Seite 1110.
- „Eigenschaft ConnectionHandle” auf Seite 1110.
- „Eigenschaft ConnectionStatus” auf Seite 1110.
- „Eigenschaft ConnectOptions” auf Seite 1110.
- „Eigenschaft DeadLetterQueueName” auf Seite 1111.
- „DefaultTransmissionQueueName-Eigenschaft” auf Seite 1111.
- „Eigenschaft Description” auf Seite 1111.
- „DistributionLists, Eigenschaft” auf Seite 1111.
- „Eigenschaft InhibitEvent” auf Seite 1111.
- „Eigenschaft IsConnected” auf Seite 1112.
- „IsOpen, Eigenschaft” auf Seite 1112.

- „Eigenschaft LocalEvent” auf Seite 1112.
- „Eigenschaft MaximumHandles” auf Seite 1112.
- „Eigenschaft MaximumMessageLength” auf Seite 1112.
- „MaximumPriority, Eigenschaft” auf Seite 1113.
- „MaximumUncommittedMessages, Eigenschaft” auf Seite 1113.
- „Name-Eigenschaft” auf Seite 1113.
- „ObjectHandle, Eigenschaft” auf Seite 1113.
- „Eigenschaft PerformanceEvent” auf Seite 1113.
- „Eigenschaft Platform” auf Seite 1113.
- „Eigenschaft 'ReasonCode'” auf Seite 1114.
- „Eigenschaft ReasonName” auf Seite 1114.
- „Eigenschaft RemoteEvent” auf Seite 1114.
- „StartStopEvent, Eigenschaft” auf Seite 1114.
- „Eigenschaft SyncPointAvailability” auf Seite 1114.
- „Eigenschaft TriggerInterval” auf Seite 1115.

Methoden

- „AccessQueue-Methode” auf Seite 1115.
- „AddDistributionList-Methode” auf Seite 1116.
- „Backout-Methode” auf Seite 1116.
- „Begin-Methode” auf Seite 1116.
- „ClearErrorCodes-Methode” auf Seite 1116.
- „Commit-Methode” auf Seite 1116.
- „Connect-Methode” auf Seite 1116.
- „Disconnect-Methode” auf Seite 1117.

Zugriff auf Eigenschaften

Auf die folgenden Eigenschaften kann zu jedem Zeitpunkt zugegriffen werden.

- „Eigenschaft AlternateUserId” auf Seite 1108.
- „CompletionCode, Eigenschaft” auf Seite 1110.
- „Eigenschaft ConnectionStatus” auf Seite 1110.
- „Eigenschaft 'ReasonCode'” auf Seite 1114.

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 zutreffen, wird von WebSphere MQ Automation Classes for ActiveX versucht, eine Verbindung zum Warteschlangenmanager herzustellen und diesen automatisch für die Abfrage zu öffnen. 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: Für den Abruf: *altuser$ = MQQueueManager*.**AlternateUserId** Zum Festlegen: *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: Zum Abrufen: *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 Stufe der WebSphere 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 WebSphere MQ-Warteschlangenmanagerobjekt.

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: Für den Abruf: *name$ = MQQueueManager*.**name**

Zum Festlegen: *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 WebSphere MQ-Wartschlangenmanager-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-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-Ursachencodes.

Syntax: Für den Abruf: *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 zur Aliasnamensumsetzung für Warteschlangenmanager finden Sie im Abschnitt Was sind Aliasnamen? In: .

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 237.

Definition

Definiert in: MQQueueManager-Klasse.

Syntax

Syntax: set queue = MQQueueManager.**AccessQueue**(Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parameter

Name$ Zeichenfolge. Name der WebSphere 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

Verbindet das MQQueueManager-Objekt über den WebSphere MQ-MQI-Client oder -Server mit einem echten Warteschlangenmanager. 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 WebSphere 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 1120.
- „Eigenschaft BackoutRequeueName” auf Seite 1120.
- „BackoutThreshold, Eigenschaft” auf Seite 1120.
- „Eigenschaft BaseQueueName” auf Seite 1120.
- „Eigenschaft 'CloseOptions'” auf Seite 1120.
- „CompletionCode, Eigenschaft” auf Seite 1121.
- „Eigenschaft ConnectionReference” auf Seite 1121.
- „CreationDateTime, Eigenschaft” auf Seite 1121.
- „Eigenschaft CurrentDepth” auf Seite 1121.
- „DefaultInputOpenOption, Eigenschaft” auf Seite 1121.
- „Eigenschaft DefaultPersistence” auf Seite 1122.
- „Eigenschaft DefaultPriority” auf Seite 1122.
- „DefinitionType, Eigenschaft” auf Seite 1122.
- „Eigenschaft DepthHighEvent” auf Seite 1122.

- „DepthHighLimit, Eigenschaft” auf Seite 1122.
- „Eigenschaft DepthLowEvent” auf Seite 1123.
- „Eigenschaft DepthLowLimit” auf Seite 1123.
- „DepthMaximumEvent, Eigenschaft” auf Seite 1123.
- „Eigenschaft DepthHighEvent” auf Seite 1122.
- „DepthHighLimit, Eigenschaft” auf Seite 1122.
- „Eigenschaft DepthLowEvent” auf Seite 1123.
- „Eigenschaft DepthLowLimit” auf Seite 1123.
- „DepthMaximumEvent, Eigenschaft” auf Seite 1123.
- „Eigenschaft Description” auf Seite 1123.
- „Eigenschaft DynamicQueueName” auf Seite 1123.
- „Eigenschaft HardenGetBackout” auf Seite 1124.
- „InhibitGet-Eigenschaft” auf Seite 1124.
- „InhibitPut-Eigenschaft” auf Seite 1124.
- „InitiationQueueName, Eigenschaft” auf Seite 1124.
- „IsOpen, Eigenschaft” auf Seite 1124.
- „MaximumDepth, Eigenschaft” auf Seite 1125.
- „Eigenschaft MaximumMessageLength” auf Seite 1125.
- „MessageDeliverySequence, Eigenschaft” auf Seite 1125.
- „ObjectHandle, Eigenschaft” auf Seite 1125.
- „Eigenschaft OpenInputCount” auf Seite 1126.
- „OpenOptions, Eigenschaft” auf Seite 1126.
- „Eigenschaft OpenOutputCount” auf Seite 1126.
- „Eigenschaft OpenStatus” auf Seite 1126.
- „ProcessName, Eigenschaft” auf Seite 1126.
- „Eigenschaft 'QueueManagerName'” auf Seite 1126.
- „Eigenschaft QueueType” auf Seite 1127.
- „Eigenschaft 'ReasonCode'” auf Seite 1127.
- „Eigenschaft ReasonName” auf Seite 1127.
- „Eigenschaft RemoteQueueManagerName” auf Seite 1127.
- „Eigenschaft RemoteQueueName” auf Seite 1127.
- „Eigenschaft 'ResolvedQueueManagerName'” auf Seite 1128.
- „Eigenschaft ResolvedQueueName” auf Seite 1128.
- „Eigenschaft RetentionInterval” auf Seite 1128.
- „Eigenschaft Scope” auf Seite 1128.
- „Eigenschaft ServiceInterval” auf Seite 1128.
- „Eigenschaft ServiceIntervalEvent” auf Seite 1128.
- „Eigenschaft Shareability” auf Seite 1129.
- „Eigenschaft TransmissionQueueName” auf Seite 1129.
- „Eigenschaft TriggerControl” auf Seite 1129.
- „Eigenschaft TriggerData” auf Seite 1129.
- „Eigenschaft TriggerDepth” auf Seite 1129.
- „TriggerMessagePriority, Eigenschaft” auf Seite 1130.

- „Eigenschaft TriggerType” auf Seite 1130.
- „Eigenschaft Usage” auf Seite 1130.

Methoden

- „ClearErrorCodes-Methode” auf Seite 1130
- „Close-Methode” auf Seite 1130
- „Get-Methode” auf Seite 1130
- „Methode Open” auf Seite 1131
- „Methode 'Put'” auf Seite 1132

Zugriff auf Eigenschaften

Wenn das Warteschlangenobjekt nicht mit einem Warteschlangenmanager verbunden ist, können Sie die folgenden Eigenschaften lesen:

- „CompletionCode, Eigenschaft” auf Seite 1121
- „Eigenschaft OpenStatus” auf Seite 1126
- „Eigenschaft 'ReasonCode'” auf Seite 1127

Sie können außerdem folgende Eigenschaften lesen und schreiben:

- „Eigenschaft AlternateUserId” auf Seite 1120
- „Eigenschaft 'CloseOptions'” auf Seite 1120
- „Eigenschaft ConnectionReference” auf Seite 1121
- „Name-Eigenschaft” auf Seite 1125
- „OpenOptions, Eigenschaft” auf Seite 1126

Wenn das Warteschlangenobjekt mit einem Warteschlangenmanager verbunden wird, können Sie alle Eigenschaften lesen.

Eigenschaften von Warteschlangenattributen

Bei den Eigenschaften, die im vorherigen Abschnitt nicht aufgeführt sind, handelt es sich um Attribute der zugrunde liegenden WebSphere 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 zu deren Verwendung berechtigt 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 (OpenSta-

tus=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'. WebSphere MQ prüft den Parameter 'OpenOptions' im Rahmen des Prozesses zum Öffnen der Warteschlange auf die Benutzerberechtigung.

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: Zum Abrufen: *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 WebSphere MQ-Warteschlangenmanagerobjekt

Syntax: Zum Festlegen: *set MQQueue***.ConnectionReference** = *ConnectionReference*

Zum Abrufen: *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.*.**EreignisDepthHigh**

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.***EreignisDepthLow**

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.***EreignisDepthMaximum**

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:

- Jeder gültige Name einer WebSphere MQ-Warteschlange

Syntax: Zum Festlegen: *MQQueue***.DynamicQueueName** = *dynamicqueuename$*

Zum Abrufen: *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: Für den Abruf: *name$ = MQQueue*.**name**

Zum Festlegen: *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 WebSphere 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: Zum Abrufen: *openincount & = MQQueue*.**OpenInputAnzahl**

OpenOptions, Eigenschaft

Lese-/Schreibzugriff. Optionen, die zum Öffnen der Warteschlange verwendet werden.

Definiert in: MQQueue-Klasse

Datentyp: Lang

Werte:

- Siehe OpenOptions (MQLONG).

Syntax: Zum Abrufen: *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: Zum Abrufen: *openoutcount & = MQQueue*.**OpenOutputAnzahl**

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: Zum Abrufen: *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 WebSphere 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-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-Ursachencodes.

Syntax: Für den Abruf: *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.***EreignisServiceInterval**

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:

- Für **MessageLength** ist die Länge der WebSphere MQ-Nachricht festgelegt
- Für **DataLength** ist die Länge der WebSphere MQ-Nachricht festgelegt
- **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:

Ein optionaler Wert einer Länge von 2 oder 4 Byte zum Steuern der maximalen Länge der WebSphere MQ-Nachricht, 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. Bei den Werten der Eigenschaften unmittelbar nach Ausführung dieser Methode handelt es sich um die Werte, die in WebSphere MQ eingereiht wurden.

Änderungen an dem MQMessage-Objekt nach Abschluss der Put-Methode haben keine Auswirkung auf die tatsächliche Nachricht in der WebSphere 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 WebSphere MQ-Nachricht dar. Sie umfasst Eigenschaften zur Einbindung des WebSphere 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.

Wenn die Puffergröße den Wert der Eigenschaft 'MaximumMessageLength' für eine WebSphere MQ-Warteschlange überschreitet, können Sie keine Nachricht in diese Warteschlange stellen.

Nach seiner Erstellung kann ein MQMessage-Objekt mithilfe der MQQueue.Put-Methode in eine WebSphere 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 ändern oder löschen, ohne dass sich dies auf die Nachricht in der WebSphere MQ-Warteschlange auswirkt. Der Warteschlangenmanager kann einige der Felder im MQMD anpassen, wenn er die Nachricht in die WebSphere MQ-Warteschlange kopiert.

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 or Set msg = New MQMessage
```

Eigenschaften

Bei den Steuerungseigenschaften handelt es sich um folgende:

Bei den Eigenschaften des Nachrichtendeskriptors handelt es sich um folgende:

Methoden

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.

Allerdings ist zu beachten, dass einige MQMD-Eigenschaften unter Umständen vom Warteschlangenmanager geändert werden, wenn die Nachricht in eine WebSphere MQ-Warteschlange eingereiht wird. In den unter MQMD aufgeführten Informationen zu den Feldern finden Sie Hinweise, wie diese Felder unter Umständen geändert werden.

Datenkonvertierung

Sie können binäre Daten an eine WebSphere MQ-Nachricht übergeben, indem Sie für die Eigenschaft 'CharacterSet' die ID des codierten Zeichensatzes des Warteschlangenmanagers (MQCCSI_Q_MGR) festlegen und die Daten als Zeichenfolge übergeben. Über die Funktion 'chr$' können Sie Nicht-Zeichendaten in die Zeichenfolge einstellen.

Die Read- und Write-Methoden führen Datenkonvertierungen durch. Sie nehmen Konvertierungen zwischen den internen ActiveX-Formaten und den WebSphere MQ-Nachrichtenformaten vor, die über die Eigenschaften 'Encoding' und 'CharacterSet' des Nachrichtendeskriptors definiert wurden. 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 WebSphere MQ-Beendigungscode zurück, der von der aktuellsten Methode oder dem letzten Eigenschaftszugriff für dieses Objekt festgelegt 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-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-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: Zum Abrufen: *actoken$ = MQMessage*.**AccountingToken**

Zum Festlegen: *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 1096.

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: Zum Abrufen: *actokenh$ = MQMessage*.**AccountingTokenHex**

Zum Festlegen: *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 1096.

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**.

Wenn CharacterSet auf **MQCCSI_Q_MGR**gesetzt ist, wird die Codepage für die aktuelle Ländereinstellung für die Zeichenkonvertierung in der Methode WriteString 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: Zum Abrufen: *correlid$ = MQMessage***.CorrelationId**; zum Setzen: *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 1096.

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: Zum Abrufen: *correlidh$ = MQMessage*.**CorrelationIdHex**

Zum Setzen: *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 1096.

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**

Zum Festlegen: *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 1096.

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: Zum Abrufen: *groupidh$ = MQMessage*.**GroupIdHex**

Zum Festlegen: *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 1096.

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: Zum Abrufen: *String$ = MQMessage*.**MessageData**

Zum Setzen: *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**

Zum Festlegen: *MQMessage*.**MessageFlags** = *messageflags &*

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: Zum Abrufen: *messageid$ = MQMessage*.**MessageId**

Zum Setzen: *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 1096.

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: Zum Abrufen: *messageidh$ = MQMessage*.**MessageIdHex**

Zum Setzen: *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 1096.

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: To get: *sequencenumber & = MQMessage*.**SequenceNumber**

Zum Festlegen: *MQMessage*.**SequenceNumber** = *sequencenumber &*

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**

Zum Festlegen: *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**

Zum Festlegen: *MQMessage*.**OriginalLength** = *originallength &*

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 Konvertierungen zwischen dem ActiveX-Datums-/Zeitformat und den Datums- und Zeitformaten durch, die in einem WebSphere MQ-MQMD verwendet werden. 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: Zum Abrufen: *datetime = MQMessage*.**PutDateTime**

Zum Festlegen: *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

Diese Methode liest 1 Byte aus dem Nachrichtendatenpuffer ab dem Zeichen, auf das über 'DataOffset' 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: MQMessage-Klasse

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

Diese Methode liest 8 Bytes aus dem Nachrichtenpuffer ab dem Byte, auf das über '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. Beachten Sie, dass die Konvertierung vom System/360-Format nicht unterstützt wird.

Definiert in: MQMessage-Klasse

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.

Diese Methode liest 4 Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt diese 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. Beachten Sie, dass die Konvertierung vom System/360-Format nicht unterstützt wird.

Definiert in: MQMessage-Klasse

Syntax: *doublev# = MQMessage.***ReadDouble4**

Methode 'ReadFloat'

Diese Methode liest 4 Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über '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. Beachten Sie, dass die Konvertierung vom System/360-Format nicht unterstützt wird.

Definiert in: MQMessage-Klasse

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*.**Lesen Int4**

Methode 'ReadLong'

Diese Methode liest 4 Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über '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: MQMessage-Klasse

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

Diese Methode liest 2 Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt sie als einen 2-Byte-Ganzzahlwert (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: MQMessage-Klasse

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'

Diese Methode liest 2 Bytes aus dem Nachrichtendatenpuffer ab dem Byte, auf das über '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: MQMessage-Klasse

Syntax: *bigint & = MQMessage*.**ReadUInt2-**

ReadUnsignedByte, Methode

Diese Methode liest 1 Byte aus dem Nachrichtendatenpuffer ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt es als 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: MQMessage-Klasse

Syntax: *integerv% = MQMessage*.**ReadUnsignedByte**

Methode ReadUTF

Diese Methode liest eine Zeichenfolge im UTF-Format aus der Nachricht ab dem Byte, auf das über 'DataOffset' verwiesen wird, und gibt sie als ActiveX-Zeichenfolge zurück. Die Zeichenfolge im UTF-Format besteht aus 2 Datenbytes, die die Länge der Zeichenfolge angeben, gefolgt von den UTF-Zeichendaten.

Wenn 'MQMessage.DataLength' kleiner als die Länge der Zeichenfolge ist, schlägt die Methode fehl.

Wenn die Methode erfolgreich ist, wird 'DataOffset' um die Länge der Zeichenfolge erhöht, 'DataLength' um die Länge der Zeichenfolge 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**(Wert &)

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 1147.

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**(*wert*)

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: Aufruf *MQMessage*.**WriteInt4**(*wert &*)

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**(*wert &*)

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: Aufruf *MQMessage*.**WriteUInt2**(*wert &*)

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

In diese Klasse sind die verschiedenen Optionen zur Steuerung des Vorgangs zum Einreihen einer Nachricht in eine WebSphere MQ-Warteschlange eingebunden.

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 1154.
- „Eigenschaft Options” auf Seite 1155.
- „Eigenschaft 'ReasonCode'” auf Seite 1155.
- „Eigenschaft ReasonName” auf Seite 1155.
- „Eigenschaft 'RecordFields'” auf Seite 1155.
- „Eigenschaft 'ResolvedQueueManagerName'” auf Seite 1156.
- „Eigenschaft ResolvedQueueName” auf Seite 1156.

Methoden

- „ClearErrorCodes-Methode” auf Seite 1156.

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-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-Ursachencodes.

Syntax: Zum Abrufen: *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 in 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. Weitere Informationen finden Sie im Handbuch *WebSphere MQ Application Programming Reference*.

Definiert in:
MQPutMessageOptions-Klasse

Datentyp:
Lang

Syntax: Zum Abrufen: *recordfields & = PutOpts*.**RecordFields**

Zum Festlegen: *PutOpts*.**RecordFields** = *recordfields &*

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: Zum Abrufen: *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.

Ist definiert in: Klasse 'MQPutMessageOptions'

Syntax: Call *PutOpts*.**ClearErrorCodes()**

Klasse 'MQGetMessageOptions'

In diese Klasse sind die verschiedenen Optionen zur Steuerung des Vorgangs zum Abrufen einer Nachricht aus einer WebSphere MQ-Warteschlange eingebunden.

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 1157
- „Eigenschaft 'MatchOptions'” auf Seite 1157
- „Eigenschaft Options” auf Seite 1157
- „Eigenschaft 'ReasonCode'” auf Seite 1157
- „Eigenschaft ReasonName” auf Seite 1158
- „Eigenschaft ResolvedQueueName” auf Seite 1158
- „WaitInterval, Eigenschaft” auf Seite 1158

Methoden

- „ClearErrorCodes-Methode” auf Seite 1158

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**

Zum Festlegen: *GetOpts*.**MatchOptions** =*matchoptions &*

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-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-Ursachencodes.

Syntax: Zum Abrufen: *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.

Ist 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 1159
- „Eigenschaft 'CloseOptions'” auf Seite 1159
- „CompletionCode, Eigenschaft” auf Seite 1159

- „Eigenschaft ConnectionReference” auf Seite 1160
- „Eigenschaft FirstDistributionListItem” auf Seite 1160
- „IsOpen, Eigenschaft” auf Seite 1160
- „OpenOptions, Eigenschaft” auf Seite 1160
- „Eigenschaft 'ReasonCode'” auf Seite 1161
- „Eigenschaft ReasonName” auf Seite 1161

Methode

- „AddDistributionListItem, Methode” auf Seite 1161
- „ClearErrorCodes-Methode” auf Seite 1161
- „Close-Methode” auf Seite 1162
- „Methode Open” auf Seite 1162
- „Methode 'Put'” auf Seite 1162

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: Zum Abrufen: *altuser$ = MQDistributionList*.**AlternateUserId**

Zum Festlegen: *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 Festlegen: *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: Zum Abrufen: *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: Zum Abrufen: *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**

Zum Festlegen: *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-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-Ursachencodes.

Syntax: Zum Abrufen: *reasonname $= MQDistributionList*.**ReasonName**

AddDistributionListItem, Methode

Erstellt ein neues MQDistributionListItem-Objekt und ordnet es dem Verteilerlistenobjekt zu. Der Parameter für den Warteschlangennamen ist obligatorisch.

Die Eigenschaft 'DistributionList' des Verteilerlistenelements wird auf die übergeordnete Verteilerliste gesetzt und die Eigenschaft 'FirstDistributionListItem' der Verteilerliste wird so gesetzt, dass sie auf das neue Verteilerlistenelement verweist.

Für das neue Verteilerlistenelement wird die Eigenschaft 'PreviousDistributionListItem' auf null gesetzt und die Eigenschaft 'NextDistributionListItem' wird entweder so gesetzt, dass sie auf das Verteilerlistenelement verweist, das zuvor das erste war, oder auf null, wenn zuvor keine Elemente in der Verteilerliste enthalten waren.

Mit dieser Methode kann kein neues Element hinzugefügt werden, solange die Verteilerliste geöffnet ist.

Definiert in:
Klasse MQDistributionList

Syntax: set distributionlistitem = *MQDistributionList***.AddDistributionListItem** (QName$, QMgrName$)

Parameter:

QName$ String. Name der WebSphere MQ-Warteschlange.

QMgrName$ String. Name des WebSphere 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, PutMsgOptionen &)

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

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 Festlegen: *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: Zum Abrufen: *accountingtokenh$ = MQDistributionListItem*.**AccountingTokenHex**

Zum Festlegen: *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: Zum Abrufen: *correlid$ = MQDistributionListItem*.**CorrelationId**

Zum Festlegen: *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: Zum Abrufen: *correlidh$ = MQDistributionListItem*.**CorrelationIdHex**

Zum Festlegen: *MQDistributionListItem*.**CorrelationIdHex** = *correlidh$*

Eigenschaft DistributionList

Schreibgeschützt. Die Verteilerliste, der dieser Verteilerlistenartikel zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionList

Syntax: Zum Abrufen: *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 Festlegen: *MQDistributionListElement*.**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 Festlegen: *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: Zum Abrufen: *groupidh$ = MQDistributionListItem*.**GroupIdHex**

Zum Festlegen: *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: Zum Abrufen: *messageid$ = MQDistributionListItem*.**MessageId**

Zum Festlegen: *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: Zum Abrufen: *messageidh$ = MQDistributionListItem*.**MessageIdHex**

Zum Festlegen: *MQDistributionListItem*.**MessageIdHex** = *messageidh$*

Eigenschaft NextDistributionListItem

Schreibgeschützt. Das nächste Objekt der Verteilerlistenelemente, das derselben Verteilerliste zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionListItem

Syntax: Zum Abrufen: *set distributionlistitem = MQDistributionListItem*.**NextDistributionListItem**

Eigenschaft PreviousDistributionListItem

Schreibgeschützt. Das vorherige Objekt der Verteilerlistenelemente, das derselben Verteilerliste zugeordnet ist.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
MQDistributionListItem

Syntax: Zum Abrufen: *set distributionlistitem = MQDistributionListItem*.**PreviousDistributionListItem**

Eigenschaft 'QueueManagerName'

Lese-/Schreibzugriff. Der Name des WebSphere MQ-Warteschlangenmanagers.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 48 Zeichen

Syntax: Zum Abrufen: *qmname$ = MQDistributionListItem*.**QueueManagerName**

Zum Festlegen: *MQDistributionListItem*.**QueueManagerName** = *qmname$*

QueueName, Eigenschaft

Lese-/Schreibzugriff. Der Name der WebSphere MQ-Warteschlange.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge von 48 Zeichen

Syntax: Zum Abrufen: *qname$ = MQDistributionListItem*.**QueueName**

Zum Festlegen: *MQDistributionListItem*.**QueueName** = *qname$*

Eigenschaft 'ReasonCode'

Schreibgeschützt. Der Ursachencode, der von der zuletzt für das übergeordnete Verteilerlistenobjekt aufgerufenen Open- oder Put-Operation gesetzt wurde.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Lang

Werte:
Siehe API-Ursachencodes.

- MQCC_OK

- MQCC_WARNING
- MQCC_FAILED

Syntax: Zum Abrufen: *reasoncode & = MQDistributionList-Element*.**ReasonCode**

Eigenschaft ReasonName

Schreibgeschützt. Der symbolische Name für den Ursachencode, beispielsweise MQRC_QMGR_NOT_AVAILABLE.

Definiert in:
Klasse MQDistributionListItem

Datentyp:
Zeichenfolge

Werte:
Siehe API-Ursachencodes.

Syntax: Zum Abrufen: *reasonname $= MQDistributionList-Element*.**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**

Fehlerbehebung

Informationen zur bereitgestellten Tracefunktion, zu häufig auftretenden Problemen und Hilfe zu ihrer Vermeidung.

Im folgenden Abschnitt wird die bereitgestellte Tracefunktion erklärt; außerdem werden Details zu häufig auftretenden Problemen und Hilfe zu ihrer Vermeidung angeboten:

- „Die Tracefunktion verwenden” auf Seite 1168
- „Vorgehensweise bei Fehlschlagen des WebSphere MQ Automation Classes for ActiveX-Scripts” auf Seite 1169
- „Ursachencodes” auf Seite 1170
- „Tool zum Ermitteln der Codeversion” auf Seite 1173

Die Tracefunktion verwenden

MQAX enthält eine Tracefunktion, mit der eine Serviceorganisation ermitteln kann, was geschieht, wenn ein Problem auftritt. Sie zeigt den Pfad an, der bei der Ausführung Ihres MQAX-Scripts verwendet wird. Wenn keine Probleme auftreten, inaktivieren Sie die Traceerstellung, um unnötiges Auslasten von Systemressourcen zu vermeiden.

Die Tracefunktion wird über drei Umgebungsvariablen gesteuert:

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

Bei Angabe von *any* für die Umgebungsvariable OMQ_TRACE wird die Tracefunktion aktiviert; Auch wenn Sie den Wert "OFF" für OMQ_TRACE angeben, ist die Traceerstellung immer noch aktiviert.

Soll die Tracefunktion inaktiviert werden, darf für OMQ_TRACE kein Wert angegeben werden.

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 (Benutzername)" 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.

Wenn die Tracefunktion aktiviert ist, ist die Ausführung von MQAX langsamer, die Leistung der ActiveX- oder WebSphere MQ-Umgebungen wird jedoch nicht beeinträchtigt. Wenn eine Tracedatei nicht mehr benötigt wird, kann sie gelöscht werden.

Soll der Status der Variablen OMQ_TRACE geändert werden, müssen Sie das MQAX-Script zuerst stoppen.

Name und Verzeichnis der Tracedatei

Der Name der Tracedatei hat das Format 'OMQnnnnn.trc'; dabei ist 'nnnnn' die ID des ActiveX-Prozesses, der zu diesem Zeitpunkt aktiv ist.

Tabelle 157. Befehle und ihre Auswirkungen

Befehl	Aktion
SET OMQ_TRACE_PATH = Laufwerk:\Verzeichnis	Setzt das Traceverzeichnis, in das die Tracedatei geschrieben werden soll.
SET OMQ_TRACE_PATH =	Entfernt alle eventuell vorhandenen Einstellungen für das Traceverzeichnis. Wird das Traceverzeichnis nicht gesetzt, wird das aktuelle Arbeitsverzeichnis (wenn ActiveX gestartet wird) verwendet.
ECHO %OMQ_TRACE_PATH%	Zeigt die aktuelle Einstellung des Traceverzeichnisses unter Windows an.
SET OMQ_TRACE = xxxxxxxx	Aktiviert die Tracefunktion. Die Tracefunktion wird aktiviert, indem nach dem Gleichheitszeichen (=) mindestens ein Zeichen angegeben wird. Beispiel: SET OMQ_TRACE=yes SET OMQ_TRACE = no. In beiden Beispielen wird die Traceerstellung aktiviert. Dies gilt nur bei einem einzelnen Fenster bzw. einer Einzelsitzung.
SET OMQ_TRACE=	Inaktiviert die Tracefunktion.
ECHO %OMQ_TRACE%	Zeigt die Inhalte der Umgebungsvariablen unter Windows an.
SET	Zeigt die Inhalte 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.

Vorgehensweise bei Fehlschlagen des WebSphere MQ Automation Classes for ActiveX-Scripts

Wenn Ihr WebSphere MQ Automation Classes for ActiveX-Script fehlschlägt, können Sie auf eine Reihe von Informationsquellen zurückgreifen.

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 ist in einer Datei des Namens 'OMQnnnnn.fdc' enthalten; dabei steht 'nnnnn' für die Nummer des ActiveX-Prozesses, der zu dem Zeitpunkt 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

In WebSphere MQ stehen abhängig von der Plattform verschiedene Fehlerprotokolle und Traceinformationen zur Verfügung. Beachten Sie das Windows NT-Anwendungsereignisprotokoll.

Ursachencodes

Zusätzlich zu den für das MQI von WebSphere MQ dokumentierten Ursachencodes können folgende Ursachencodes auftreten. Informationen zu weiteren Codes finden Sie im WebSphere MQ-Anwendungsereignisprotokoll.

Tabelle 158. Ursachencodes und ihre Bedeutung

Ursachencode	Beschreibung
MQRC_LIBRARY_LOAD_ERROR (6000)	Mindestens eine WebSphere MQ-Bibliothek konnte nicht geladen werden. Prüfen Sie, ob sich alle WebSphere MQ-Bibliotheken im korrekten Suchpfad auf dem von Ihnen verwendeten System befinden. Vergewissern Sie sich beispielsweise, dass sich die Verzeichnisse mit den WebSphere MQ-Bibliotheken in PATH befinden.
MQRC_CLASS_LIBRARY_ERROR (6001)	Einer der WebSphere MQ-Aufrufe 'classlibrary' hat einen unerwarteten Ursachencode/Beendigungscode 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 dem 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 wurde ein Wert außerhalb des gültigen Bereichs verwendet; Beispiel: msg.WriteByte (240).
MQRC_PACKED_DECIMAL_ERROR (6004)	Es wurde versucht, eine gepackte Dezimalzahl aus dem Nachrichtenpuffer zu lesen, doch 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 entsprechenden Optionen für das Öffnen (**OpenOptions**); es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich, doch ein Schließen wurde verhindert. Setzen Sie die Eigenschaft **OpenOptions** so, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist. Ein Schließen wurde verhindert, da die Warteschlange für eine exklusive Eingabe geöffnet ist und durch ein Schließen andere die Möglichkeit hätten, auf die Warteschlange zuzugreifen.

Tabelle 158. Ursachencodes und ihre Bedeutung (Forts.)

Ursachencode	Beschreibung
MQRC_REOPEN_INQUIRE_ERROR (6101)	Ein geöffnetes Objekt verfügt nicht über die entsprechenden Optionen für das Öffnen (OpenOptions); es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich, doch ein Schließen wurde verhindert. Setzen Sie die Eigenschaft 'OpenOptions' so, dass MQOO_INQUIRE enthalten ist. Ein Schließen wurde verhindert, da ein oder mehrere Merkmale des Objekts dynamisch überprüft werden müssen, bevor es geschlossen wird, und MQOO_INQUIRE noch nicht in der Eigenschaft 'OpenOptions' enthalten ist.
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Ein geöffnetes Objekt verfügt nicht über die entsprechenden Optionen für das Öffnen (OpenOptions); es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich, doch ein Schließen wurde verhindert. Setzen Sie die Eigenschaft 'OpenOptions' so, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist. Ein Schließen wurde verhindert, da die Warteschlange mit MQOO_SAVE_ALL_CONTEXT geöffnet wurde und zuvor ein Abruf mit Löschen ausgeführt wurde. Dadurch wurden Statusinformationen, die beibehalten wurden, der geöffneten Warteschlange zugeordnet, und diese Informationen würden beim Schließen verloren gehen.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Ein geöffnetes Objekt verfügt nicht über die entsprechenden Optionen für das Öffnen (**OpenOptions**); es ist mindestens eine weitere Option erforderlich. Es ist ein implizites erneutes Öffnen erforderlich, doch ein Schließen wurde verhindert. Setzen Sie die Eigenschaft **OpenOptions** so, dass alle möglichen Fälle abgedeckt sind und kein implizites erneutes Öffnen erforderlich ist. Ein Schließen wurde verhindert, da es sich bei der Warteschlange um eine lokale Warteschlange des Definitionstyps MQQDT_TEMPORARY_DYNAMIC handelt, die beim Schließen gelöscht werden würde.
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. Setzen Sie die Eigenschaft 'OpenOptions' so, 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.
MQRC_STRUCID_ERROR (6107)	Die Struktur der ID für das nächste Nachrichtenelement, die von den 4 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. Dies kann darauf zurückzuführen sein, dass explizite Deklarationen für WebSphere MQ-Objekte aus VBA als Parameter für Aufrufe verwendet wurden (so kann beispielsweise 'dim msg as Object' verwendet werden, die Verwendung von 'dim msg as MqMessage' kann dagegen zu Problemen führen). In Excel beispielsweise (wenn eine Warteschlange definiert und gesetzt ist) wird bei Verwendung des Befehls 'set dim msg as MqMessageq.put msg' der Ursachencode MQRC_NULL_POINTER zurückgegeben. In Visual Basic hingegen tritt bei diesem Befehl kein Problem auf.
MQRC_NO_CONNECTION_REFERENCE (6109)	Das Objekt **MQQueue** hat keine Verbindung mehr zu **MQQueueManager**. Dieser Ursachencode tritt auf, wenn die Verbindung zu **MQQueueManager** getrennt wurde. Löschen Sie das Objekt **MQQueue**.
MQRC_NO_BUFFER (6110)	Es ist kein Puffer verfügbar. Für ein **MQMessage**-Objekt kann kein Puffer zugeordnet werden; dies deutet auf eine interne Inkonsistenz des Objektstatus hin, die nicht auftreten sollte.

Tabelle 158. Ursachencodes und ihre Bedeutung (Forts.)

Ursachencode	Beschreibung
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 eine **CorrelationId** und für eine **MessageId** 32 ist eine korrekte Länge für ein **AccountingToken**
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.
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 MQQueueManager-Objekt, auf das verwiesen wird und das nicht verbunden ist.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	Der MQPutMessageOptions-Kontextverweis verweist nicht auf ein gültiges MQQueue-Objekt. Das Objekt wurde zuvor gelöscht.
MQRC_CONTEXT_OPEN_ERROR (6122)	Der MQPutMessageOptions-Kontextverweis verweist auf ein MQQueue-Objekt, das nicht geöffnet werden konnte, um einen Kontext zu erstellen. Eine mögliche Ursache ist, dass für das MQQueue-Objekt keine geeigneten 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 ist die Länge für die Aufnahme der festen Felder und aller Offset-Daten nicht ausreichend.
MQRC_NOT_CONNECTED (6124)	Eine Methode ist fehlgeschlagen, da eine zum Warteschlangenmanager erforderliche Verbindung nicht verfügbar war und und eine Verbindung nicht implizit hergestellt werden kann.
MQRC_NOT_OPEN (6125)	Eine Methode ist fehlgeschlagen, da ein WebSphere MQ-Objekt nicht offen war und ein implizites Öffnen nicht möglich ist.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	Eine Verteilerliste (MQDistributionList) konnte nicht geöffnet werden, da keine MQDistributionListItem-Objekte in der Verteilerliste enthalten sind. Korrekturmaßnahme: Fügen Sie der Verteilerliste mindestens ein MQDistributionListItem-Objekt hinzu.
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. Korrekturmaßnahme: Öffnen Sie das Objekt mit den entsprechenden Optionen für das Öffnen und wiederholen Sie den Vorgang.

Tabelle 158. Ursachencodes und ihre Bedeutung (Forts.)	
Ursachencode	**Beschreibung**
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

Vielleicht möchte das IBM Service-Team 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, 2024. 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 (WebSphere MQ Automation Classes for ActiveX)” auf Seite 1094.

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 Einführung in das WebSphere MQ Administration Interface (MQAI). Die MQAI-Klasse ActiveX MQBag kapselt die von der MQAI unterstützten Datenbehälter auf eine Weise, die in jeder Sprache verwendet werden kann, die die Erstellung von COM-Objekten unterstützt, z. B. Visual Basic, C + +, Java und andere ActiveX .

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 1095.

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-Execute-Methode sendet die Daten aus den Behältern als PCF-Nachricht an einen WebSphere MQ-Warteschlangenmanager und sammelt die Antworten.

Weitere Informationen zur Klasse 'MQBag' und ihren Eigenschaften und Methoden finden Sie unter „Klasse 'MQBag'” auf Seite 1174.

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 durch Auswahl der entsprechenden Anforderungs- und Antwortwarteschlangen an alle Warteschlangenmanager im WebSphere 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 1175.
- „Eigenschaft 'Count'” auf Seite 1176.
- „Eigenschaft Options” auf Seite 1176.

MQBag-Methoden

Informationen zu den Methoden der MQBag-Objekte finden Sie unter den folgenden Links:

- „Methode 'Add'” auf Seite 1177.
- „AddInquiry, Methode” auf Seite 1178.
- „Methode 'Clear'” auf Seite 1178.
- „Methode 'Execute'” auf Seite 1178.
- „Methode 'FromMessage'” auf Seite 1179.
- „Methode 'ItemType'” auf Seite 1179.
- „Remove, Methode” auf Seite 1180.
- „Selector, Methode” auf Seite 1181.
- „Methode 'ToMessage'” auf Seite 1181.
- „Truncate, Methode” auf Seite 1182.

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: setzt sich aus dem WebSphere MQ-Ursachencode für den festgestellten Fehler und einem COM-Facility-Code zusammen. Das Facility-Feld gibt bei COM standardmäßig den Bereich an, aus dem der Fehler stammt. Bei Fehlern die von WebSphere MQ erkannt werden, lautet dieser 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 für den WebSphere MQ-Ursachencode.

Wie Sie auf die Fehlerinformationen zugreifen können, hängt von Ihrer Scriptsprache ab. In Visual Basic beispielsweise werden die Informationen im Err-Objekt zurückgegeben, den WebSphere MQ-Ursachencode erhält man durch Subtraktion der Konstante 'vbObjectError' von 'Err.Number'.

ReasonCode = Err.Number - vbObjectError

Wenn die MQBag-Execute-Methode 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 die Beendigungs-und Fehlerursachencodes, wie in Definitionen der programmierbaren Befehlsformate beschrieben.

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 Referenzinformationen zu Programmable Command Formats (PCF)).

Format

Item (Selector, ItemIndex, Value)

Parameter

Selector **(VARIANT) - input**
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) - input**
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) - input/output**
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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Count (*Selector, Value*)

Parameter

***Selector* (VARIANT) - input**
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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Options (*Options*)

Parameter

***Options* (LONG) - input/output**
Die Datenbehälteroptionen.

Anmerkung: Die Behälteroptionen müssen festgelegt werden, ***bevor*** dem Behälter Datenelemente hinzugefügt werden oder in dem Behälter Datenelemente 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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Add (*Value, Selector*)

Parameter

***Value* (VARIANT) - input**
Ganzzahl- oder Zeichenfolgewert des Datenelements.

***Selector* (VARIANT) - input**
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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

AddInquiry *(Inquiry)*

Parameter

***Inquiry* (LONG) - input**
Selektor für das WebSphere 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 Referenzinformationen zu Programmable Command Formats (PCF).

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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Führen Sie *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*aus.

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 Referenzinformationen zu Programmable Command Formats (PCF).

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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

ItemType *(Selector, ItemIndex, ItemType)*

Parameter

***Selector* (VARIANT) - input**
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) - input**
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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Remove *(Selector, ItemIndex)*

Parameter

***Selector* (VARIANT) - input**
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) - input**
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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Selector ***(Selector, ItemIndex, OutSelector)***

Parameter

Selector **(VARIANT) - input**
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) - input**
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 Referenzinformationen zu Programmable Command Formats (PCF).

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 Referenzinformationen zu Programmable Command Formats (PCF).

Format

Truncate *(ItemCount)*

Parameter

ItemCount **(LONG) - input**
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)
```

Informationen zu den Einführungsbeispielen für die WebSphere MQ Automation Classes for ActiveX

In diesem Anhang werden die Einführungsbeispiele für die WebSphere MQ Automation Classes for ActiveX und ihre Verwendung erläutert.

In WebSphere MQ for Windows stehen folgende Visual Basic-Beispielprogramme zur Verfügung:

- 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.

In demselben Verzeichnis finden Sie auch Beispiele 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 Beispiele zeigen die Verwendung von WebSphere MQ Automation Classes for ActiveX, um folgende Aktionen auszuführen:

- 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 1183

„Fehlerbehandlung in den Beispielen” auf Seite 1183

Ausführen, ActiveX-Einführungsbeispiele

Vergewissern Sie sich vor der Ausführung der Einführungsbeispiele für WebSphere MQ Automation Classes for ActiveX, dass ein Standardwarteschlangenmanager aktiv ist und die erforderlichen Warteschlangendefinitionen erstellt wurden. Ausführliche Informationen zum Erstellen und Ausführen eines Warteschlangenmanagers sowie zum Erstellen einer Warteschlange finden Sie unter Verwaltung . Das Beispiel verwendet die Warteschlange SYSTEM.DEFAULT.LOCAL.QUEUE , die auf jedem normalerweise eingerichteten WebSphere 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-Einführungsbeispielen für Microsoft Basic Version 4 oder höher finden Sie im Abschnitt „Beispielprogramm MQAXTRIV ausführen” auf Seite 1184.

Informationen zu einem Beispielprogramm, mit dem Sie Eigenschaften und Methoden von Warteschlangenmanagern und Warteschlangenobjekten durchsuchen können, finden Sie unter „Beispiel MQAXCLSS starten” auf Seite 1185.

Informationen zum Beispiel MQAXDLST finden Sie unter „MQAXDLST-Beispiel” auf Seite 1186.

Informationen zur Ausführung des MQAX-Einführungsbeispiels für Microsoft Excel 95 oder höher, MQAXTRIV.XLS, finden Sie im Abschnitt „Beispiel MQAXTRIV.XLS ausführen” auf Seite 1186.

Informationen zum Ausführen der Bankdemo mit MQAX.XLS finden Sie unter „Bankdemo mit MQAX.XLS ausführen” auf Seite 1186.

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 1186.

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 Version 4 (oder höher)
- Microsoft Excel 95 (oder höher)
- Einen Web-Browser

Außerdem benötigen Sie Folgendes:

- Einen aktiven WebSphere MQ-Warteschlangenmanager.
- Eine bereits definierte WebSphere MQ-Warteschlange.

Fehlerbehandlung in den Beispielen

Die meisten Beispiele im Paket der WebSphere MQ Automation Classes for ActiveX geben nur wenige oder keine Hinweise zur Fehlerbehandlung. Weitere Informationen zur Fehlerbehandlung finden Sie unter „Fehlerbehandlung” auf Seite 1099.

Beispielprogramm MQAXTRIV ausführen

1. Starten Sie den Warteschlangenmanager.
2. Wählen Sie in Windows Explorer oder File Manager das Symbol für das Beispiel, MQAXTRIV.VBP (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 a WebSphere MQ message to
'* and from a WebSphere 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   '* get message option

Dim GetOptions As MQGetMessageOptions   '* put 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 "WebSphere MQ Completion Code = " & MQSess.CompletionCode
    Print "WebSphere 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 in Windows Explorer doppelt auf das Dokumentsymbol klicken oder im Dateimenü in Visual Basic 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-Einführungsbeispiel für Microsoft Excel 95 oder höher

In diesem Abschnitt wird erläutert, wie das MQAX-Einführungsbeispiel für Microsoft Excel 95 oder höher, 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 WebSphere MQ-MQSC-Befehl BANK.TST aus. Dabei werden die erforderlichen IBM WebSphere 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 Beispiel handelt es sich um die Client-IBM WebSphere MQ-Demonstration.
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 Microsoft Internet Explorer (nicht jedoch Netscape Navigator) ist ein kompatibler Web-Browser.

Ausführen, HTML-Beispiel

Dieses Beispiel zeigt, wie MQAX über VBScript und JavaScript aufgerufen werden kann.

1. Starten Sie den Warteschlangenmanager.
2. Öffnen Sie in dem mit ActiveX kompatiblen Web-Browser die Datei MQAXTRIV.HTM.

   Zu diesem Zweck können Sie doppelt auf das Dateisymbol in Windows Explorer klicken oder im Dateimenü Ihres mit ActiveX kompatiblen Web-Browsers 'File - Open' (Datei - Öffnen) auswählen.
3. Gehen Sie entsprechend den Anweisungen am Bildschirm vor.

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:

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 zu geplanten Programmierschnittstellen, die es dem Kunden ermöglichen, Programme zum Abrufen der Services von IBM WebSphere MQzu schreiben.

Diese Informationen können jedoch auch Angaben über Diagnose, Bearbeitung und Optimierung enthalten. Die Informationen zu Diagnose, Bearbeitung und Optimierung sollten Ihnen bei der Fehlerbehebung für die Anwendungssoftware helfen.

Wichtig: Verwenden Sie diese Diagnose-, Änderungs-und Optimierungsinformationen nicht als Programmierschnittstelle, da sie Änderungen unterliegen.

Marken

IBM, das IBM Logo, ibm.com, sind Marken der IBM Corporation in den USA und/oder anderen Ländern. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

Microsoft und Windows sind 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: