9.3

Desarrollo de aplicaciones para IBM MQ

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información en “Avisos” en la página 1329.

Esta edición se aplica a la versión 9 release 3 de IBM® MQ y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

Cuando envía información a IBM, otorga a IBM un derecho no exclusivo para utilizar o distribuir la información de la forma que considere adecuada, sin incurrir por ello en ninguna obligación con el remitente.

© Copyright International Business Machines Corporation 2007, 2024.

Contenido

Desarrollo de aplicaciones 5
Conceptos de desarrollo de aplicaciones 7
Acciones que pueden realizar las aplicaciones 7
Aplicaciones, nombres de aplicación e instancias de aplicación 9
Programas de aplicación que utilizan la MQI 10
Utilización de conexiones cliente para conectarse a varios gestores de colas IBM MQ 11
Desarrollo de aplicaciones cliente flexibles y escalables 15
Aplicaciones orientadas a objetos 16
Mensajes de IBM MQ 18
Preparación y ejecución de aplicaciones Microsoft Transaction Server 50
Consideraciones acerca del diseño de las aplicaciones IBM MQ 51
Especificación del nombre de aplicación en los lenguajes de programación admitidos 54
Técnicas de diseño para mensajes 60
Consideraciones sobre el diseño de aplicaciones y el rendimiento 62
Técnicas de diseño de aplicaciones avanzadas 64
Consideraciones de diseño y rendimiento para aplicaciones de IBM i 65
Consideraciones acerca del diseño de las aplicaciones Linux on Power Systems - Little Endian 67
Consideraciones de diseño y rendimiento para aplicaciones de z/OS 67
IMS y las aplicaciones puente IMS en IBM MQ for z/OS 72
Desarrollo de aplicaciones JMS/Jakarta Messaging y Java 84
Utilización de IBM MQ classes for JMS/Jakarta Messaging 85
Utilización de IBM MQ classes for Java 356
Utilización del adaptador de recursos de IBM MQ 448
Utilización de IBM MQ y WebSphere Application Server juntos 515
Utilización del paquete de cabeceras de IBM MQ 532
Configuración de IBM MQ en IBM i con Java y JMS 535
Desarrollo de aplicaciones Java utilizando un repositorio de Maven 542
Desarrollo de aplicaciones C++ 543
Programas de ejemplo C++ 546
consideraciones relativas a C++ 550
Mensajería en C++ 554
Creación de programas IBM MQ en C++ 561
Desarrollo de aplicaciones .NET 571
Instalación del IBM MQ classes for .NET 572
Instalación del IBM MQ classes for .NET Framework 579
Opciones para conectar IBM MQ classes for .NET a un gestor de colas 580
Aplicaciones de ejemplo para .NET 580
Configuración del gestor de colas para aceptar conexiones de cliente TCP/IP 583
Transacciones distribuidas en .NET 583
Escritura y despliegue de programas de IBM MQ .NET 596
Desarrollo de aplicaciones XMS .NET 633
Estilos de mensajería soportados por XMS 633
Modelo de objeto XMS 634
El modelo de mensaje XMS 637
Instalación del IBM MQ classes for XMS .NET 637
Configuración del entorno de servidor de mensajería 642
Utilización de las aplicaciones de ejemplo XMS 647
Cómo escribir aplicaciones de XMS 650
Escritura de aplicaciones de XMS .NET 669
Cómo trabajar con objetos administrados de XMS .NET 675
Cómo evitar que las aplicaciones utilicen una versión más nueva de XMS 682
Protección de comunicaciones para aplicaciones XMS 683

Mensajes de XMS ... 686
Desarrollo de aplicaciones cliente AMQP ... 695
MQ Light, Apache Qpid JMSy AMQP (Advanced Message Queuing Protocol) ... 698
Soporte de AMQP 1.0 ... 699
Soporte punto a punto en canales AMQP ... 701
Correlación de campos de mensaje AMQP e IBM MQ ... 702
Fiabilidad de entrega de mensajes ... 709
Topologías para clientes AMQP con IBM MQ ... 713
Propiedades de control de escucha AMQP de IBM MQ ... 720
Desarrollo de aplicaciones REST con IBM MQ ... 721
Mensajería utilizando la REST API ... 723
Desarrollo de aplicaciones MQI con IBM MQ ... 736
Archivos de definición de datos de IBM MQ ... 736
Desarrollo de una aplicación procedimental para encolamientos ... 740
Desarrollo de aplicaciones procedimentales cliente ... 936
Salidas de usuario, salidas de API y servicios instalables de IBM MQ ... 960
Creación de una aplicación procedimental ... 1025
Tratamiento de errores en un programa procedimental ... 1064
Programación de Multicast ... 1070
Codificación en C ... 1077
Codificación en Visual Basic ... 1079
Desarrollo en COBOL ... 1080
Codificación en lenguaje ensamblador System/390 (interfaz de cola de mensajes) ... 1081
Desarrollo de programas IBM MQ en RPG (solo IBM i) ... 1084
Codificación en PL/I (solo z/OS) ... 1084
Utilización de programas procedimentales de ejemplo de IBM MQ ... 1085
Desarrollo de aplicaciones para Managed File Transfer ... 1250
Especificación de programas que se van a ejecutarse con MFT ... 1250
Utilización de Apache Ant con MFT ... 1253
Personalización de MFT con salidas de usuario ... 1258
Control de MFT colocando mensajes en la cola de mandatos de agente ... 1272
Desarrollo de aplicaciones para MQ Telemetry ... 1272
Programas de ejemplo de IBM MQ Telemetry Transport ... 1273
Conceptos sobre la programación del cliente de MQTT ... 1275
Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ ... 1297
Introducción al canal personalizado de IBM MQ para WCF con .NET ... 1297
Utilización de canales personalizados IBM MQ para WCF ... 1302
Utilización de los ejemplos de WCF ... 1322

Avisos ... 1329
Información acerca de las interfaces de programación ... 1330
Marcas registradas ... 1331

Desarrollo de aplicaciones para IBM MQ

Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

Novedades en el desarrollo de aplicaciones para IBM MQ?

Para obtener más información sobre cómo desarrollar aplicaciones para IBM MQ, visite IBM Developer:

- IBM MQ Developer Essentials *(aprenda los conceptos básicos, ejecute una demo, codifique una aplicación, realice guías de aprendizaje más avanzadas)*
- IBM MQ Downloads for Developers *(incluidas las ediciones de desarrollador gratuitas y las versiones de prueba)*

Es posible que también le resulte más fácil desarrollar las aplicaciones si está familiarizado con los conceptos descritos en las secciones siguientes:

- “Conceptos de desarrollo de aplicaciones” en la página 7
- “Consideraciones acerca del diseño de las aplicaciones IBM MQ” en la página 51

Soporte para lenguajes e infraestructuras orientados a objetos

IBM MQ proporciona un soporte principal para las aplicaciones desarrolladas en los lenguajes e infraestructuras siguientes:

- JMS
- Java
- C++
- .NET

Consulte también “Aplicaciones orientadas a objetos” en la página 16.

.NET admite las aplicaciones desarrolladas en muchos lenguajes. Para ilustrar el uso de las clases IBM MQ para .NET para acceder a colas de IBM MQ, la documentación del producto MQ contiene información para los lenguajes siguientes:

- C# código de ejemplo y aplicaciones de ejemplo
- Aplicaciones de ejemplo C++
- Visual Basic aplicaciones de ejemplo

Consulte “Escritura y despliegue de programas de IBM MQ .NET” en la página 596.

IBM MQ da soporte a .NET Core para aplicaciones en entornos Windows desde IBM MQ 9.1.1 y para aplicaciones en entornos Linux® desde IBM MQ 9.1.2. Para obtener más información, consulte “Instalación del IBM MQ classes for .NET” en la página 572.

ALW IBM MQ también da soporte a los clientes AMQP que implementan el protocolo OASIS AMQP 1.0 .

MQ Light, Apache Los clientes Qpid como Apache Qpid Proton y las API Apache Qpid JMS se basan en este protocolo.

Las API de MQ Light están disponibles en IBM MQ Light.

Los clientes Qpid de Apache están disponibles en QPid Proton.

Los enlaces de lenguaje siguientes se proporcionan tal cual:

- un enlace Ir

© Copyright IBM Corp. 2007, 2024

- una implementación de API JavaScript que funciona con aplicaciones Node.js

Soporte para API REST de programación

IBM MQ proporciona soporte para las siguientes API de REST de programación para enviar y recibir mensajes:

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

Consulte "Desarrollo de aplicaciones REST con IBM MQ" en la página 721, y también la guía de aprendizaje Cómo empezar con la API REST de mensajería de IBM MQ en el área IBM MQ de IBM Developer. Esta guía de aprendizaje incluye ejemplos en los lenguajes siguientes, proporcionados tal cual, para su uso con IBM MQ messaging REST API:

- Ejemplo de go que utiliza la API REST de mensajería de MQ
- Ejemplo de Node.js utilizando el módulo HTTPS
- Ejemplo de Node.js con el módulo Promise

Soporte para lenguajes de programación procedimentales

IBM MQ proporciona soporte para las aplicaciones desarrolladas en los lenguajes de programación de procedimiento siguientes:

- C
- Windows Visual Basic (solo sistemas Windows)
- COBOL
- z/OS Ensamblador (solo IBM MQ for z/OS)
- z/OS PL/I (solo IBM MQ for z/OS)
- IBM i RPG (solo IBM MQ for IBM i)

Estos lenguajes utilizan la interfaz de cola de mensajes (MQI) para acceder a los servicios de colocación de mensajes en colas. Consulte "Desarrollo de aplicaciones MQI con IBM MQ" en la página 736. Tenga en cuenta que el modelo de objeto de IBM MQ, utilizado por los lenguajes orientados a objetos y las infraestructuras, proporciona funciones adicionales que no están disponibles para los lenguajes de procedimiento que utilizan la MQI.

Especificación del nombre de aplicación

ALW

Antes de IBM MQ 9.1.2, podría especificar un nombre de aplicación en aplicaciones cliente Java o JMS. A partir de IBM MQ 9.1.2, también puede especificar el nombre de aplicación en lenguajes de programación adicionales. Para obtener más información, consulte "Especificación del nombre de aplicación en los lenguajes de programación admitidos" en la página 54.

Tareas relacionadas

"Desarrollo de aplicaciones para MQ Telemetry" en la página 1272

"Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ" en la página 1297

El canal personalizado de Microsoft Windows Communication Foundation (WCF) para IBM MQ envía y recibe mensajes entre clientes y servicios WCF.

Referencia relacionada

"Desarrollo de aplicaciones para Managed File Transfer" en la página 1250
Especifique los programas que se van a ejecutar con Managed File Transfer, utilice Apache Ant con Managed File Transfer, personalice Managed File Transfer con salidas de usuario y controle Managed File Transfer colocando los mensajes en la cola de mandatos de agente.

Conceptos de desarrollo de aplicaciones

Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

Para obtener más información sobre los tipos de aplicación que puede escribir para IBM MQ, consulte "Desarrollo de aplicaciones para IBM MQ" en la página 5 y "Acciones que pueden realizar las aplicaciones" en la página 7.

Conceptos relacionados

"Consideraciones acerca del diseño de las aplicaciones IBM MQ" en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

Acciones que pueden realizar las aplicaciones

Puede desarrollar aplicaciones para enviar y recibir mensajes que necesite para dar soporte a sus procesos de negocio. También puede desarrollar aplicaciones para gestionar los gestores de colas y recursos relacionados.

Acciones que pueden realizar sus aplicaciones en IBM MQ for Multiplatforms

En Multiplatforms, puede escribir aplicaciones que realicen las acciones siguientes:

- Enviar mensajes a otras aplicaciones que se ejecuten bajo los mismos sistemas operativos. Las aplicaciones pueden estar en el mismo sistema o en otro.
- Enviar mensajes a aplicaciones que se ejecutan en otras plataformas IBM MQ.
- Utilice la colocación de mensajes en colas en CICS para los sistemas siguientes:
  -

TXSeries para AIX
  -

IBM i
  -

Windows
- Utilice la colocación de mensajes en colas en Encina para los sistemas siguientes:
  -

AIX
  -

Windows
- Utilice la colocación de mensajes en colas en Tuxedo para los sistemas siguientes:
  -

AIX
  - AT&T
  -

Windows
- Utilizar IBM MQ como gestor de transacciones, coordinando actualizaciones realizadas por gestores de recursos externos dentro de las unidades de trabajo de IBM MQ. Se soportan los gestores de recursos externos siguientes y cumplen con la interfaz de X/Open XA
  - Db2

  - Informix
  - Oracle
  - Sybase
- Procesar varios mensajes juntos como una única unidad de trabajo que se puede confirmar o restituir.
- Ejecutarse desde un entorno IBM MQ completo o desde un entorno de cliente IBM MQ.

Acciones que pueden realizar sus aplicaciones en IBM MQ for z/OS

z/OS

En z/OS, puede escribir aplicaciones que realicen las acciones siguientes:

- Utilizar colas de mensajes dentro de CICS o IMS.
- Enviar mensajes entre aplicaciones por lotes, CICS y de IMS, seleccionando el entorno más adecuado para cada función.
- Enviar mensajes a aplicaciones que se ejecutan en otras plataformas IBM MQ.
- Procesar varios mensajes juntos como una única unidad de trabajo que se puede confirmar o restituir.
- Enviar mensajes e interactuar con aplicaciones IMS por medio del puente IMS.
- Participar en unidades de trabajo coordinadas por RRS.

Cada entorno dentro de z/OS tiene sus propias características, ventajas y desventajas. La ventaja de IBM MQ for z/OS es que las aplicaciones no están sujetas a ningún entorno, sino que se pueden distribuir para aprovechar las ventajas de cada entorno. Por ejemplo, puede desarrollar interfaces de usuario final utilizando TSO o CICS, puede ejecutar módulos de proceso intensivo en el proceso por lotes de z/OS y puede ejecutar aplicaciones de base de datos en IMS o CICS. En todos los casos, las diversas partes de la aplicación se pueden comunicar utilizando mensajes y colas.

Los diseñadores de aplicaciones de IBM MQ deben tener en cuenta las diferencias y limitaciones impuestas por estos entornos. Por ejemplo:

- IBM MQ proporciona recursos que permiten la intercomunicación entre gestores de colas (esto se conoce como *colas distribuidas*).
- Los métodos de confirmación y restitución de cambios difieren entre los entornos de lotes y CICS.
- IBM MQ for z/OS proporciona soporte en el entorno IMS para programas de proceso de mensajes en línea (MPP), programas de vía de acceso rápida interactiva (IFP) y programas de proceso de mensajes por lotes (BMP). Si escribe programas DL/I por lotes, siga las directrices proporcionadas en temas como “Creación de aplicaciones por lotes de z/OS” en la página 1050 y “Consideraciones acerca de los programas z/OS por lotes” en la página 750 para programas por lotes de z/OS.
- Aunque pueden existir varias instancias de IBM MQ for z/OS en un único sistema z/OS, una región CICS se puede conectar con un solo gestor de colas al mismo tiempo. No obstante, es posible conectar más de una región CICS con el mismo gestor de colas. En los entornos de lotes IMS y z/OS, los programas se pueden conectar con más de un gestor de colas.
- IBM MQ for z/OS permite que un grupo de gestores de colas pueda compartir colas locales, proporcionando un rendimiento y una disponibilidad mejorados. Estas colas se llaman *colas compartidas* y los gestores de colas forman un *grupo de compartición de colas*, que puede procesar mensajes en las mismas colas compartidas. Las aplicaciones por lotes pueden conectarse a uno de varios gestores de colas dentro de un grupo de compartición de colas especificando el nombre del grupo de compartición de colas, en lugar de un nombre de gestor de colas concreto. Esto se conoce como *conexión por lotes de grupo*, o simplemente *conexión de grupo*. Consulte Colas compartidas y grupos de compartición de colas.

z/OS Las diferencias entre los entornos admitidos, así como sus limitaciones, se explican con mayor detalle en “Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911.

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 7

Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

"Consideraciones acerca del diseño de las aplicaciones IBM MQ" en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

"Desarrollo de una aplicación procedimental para encolamientos" en la página 740
Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

"Desarrollo de aplicaciones procedimentales cliente" en la página 936
Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

"Utilización de IBM MQ classes for JMS/Jakarta Messaging" en la página 85
IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

"Utilización de IBM MQ classes for Java" en la página 356
Utilice IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

"Desarrollo de aplicaciones .NET" en la página 571
IBM MQ classes for .NET permitir que las aplicaciones .NET se conecten a IBM MQ como un IBM MQ MQI client o que se conecten directamente a un servidor IBM MQ .

"Desarrollo de aplicaciones C++" en la página 543
IBM MQ proporciona clases C++ equivalentes a los objetos IBM MQ y algunas clases equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

"Creación de una aplicación procedimental" en la página 1025
Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

Tareas relacionadas

"Utilización de programas procedimentales de ejemplo de IBM MQ" en la página 1085
Estos programas de ejemplo se escriben en lenguajes de procedimiento y muestran los usos típicos de la interfaz de cola de mensajes (MQI). Programas IBM MQ en distintas plataformas.

"Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ" en la página 1297
El canal personalizado de Microsoft Windows Communication Foundation (WCF) para IBM MQ envía y recibe mensajes entre clientes y servicios WCF.

Seguridad

Multi Aplicaciones, nombres de aplicación e instancias de aplicación

Antes de empezar a diseñar y escribir las aplicaciones, familiarícese con los conceptos básicos de aplicaciones, nombres de aplicación e instancias de aplicación.

Aplicaciones

Multi

Las conexiones a un gestor de colas se consideran que proceden de la misma *aplicación* si proporcionan el mismo *nombre de aplicación*. El nombre de la aplicación se muestra cómo el atributo de APPLTAG del mandato DISPLAY CONN(*) TYPE CONN.

Notas:

1. Para las aplicaciones que utilizan una versión de IBM MQ client anterior a IBM MQ 9.1.2, el nombre de la aplicación se establece automáticamente mediante IBM MQ client. Su valor depende del lenguaje de programación de aplicaciones y de la plataforma en la que se ejecuta la aplicación. Consulte PutApplName para obtener más información.
2. Para las aplicaciones IBM MQ client que utilizan un IBM MQ client en IBM MQ 9.1.2 o posterior, es posible establecer el nombre de la aplicación en un valor específico. En la mayoría de los casos, esto no requiere cambios en el código de la aplicación o una necesidad de recompilar la aplicación. En "Utilización del nombre de aplicación en lenguajes de programación admitidos" en la página 55 encontrará más información.

Instancias de aplicación

Las conexiones se subdividen más en *instancias de aplicación*. Una instancia de una aplicación es un conjunto de conexiones estrechamente relacionadas que proporcionan una 'unidad de ejecución' para dicha aplicación. Normalmente, se trata de un único proceso de sistema operativo, que puede tener distintas hebras y conexiones IBM MQ asociadas.

En IBM MQ for Multiplatforms, una instancia de aplicación se asocia a una Etiqueta de conexión. El gestor de colas asocia automáticamente las nuevas conexiones a una instancia de aplicación existente, cuando puede ver que están relacionados.

Notas:

- Si se utilizan conexiones de cliente, estos procesos se podrían conectar al gestor de colas a través de uno o más canales en ejecución.
- En las aplicaciones JMS, una instancia de aplicación se correlaciona con una conexión JMS específica y todas las sesiones JMS asociadas.

Las instancias de aplicación son particularmente importantes en IBM MQ for Multiplatforms cuando se utiliza el equilibrado automático de aplicaciones del clúster uniforme. En las plataformas IBM MQ for Multiplatforms, puede ver las instancias de aplicación conectadas actualmente utilizando el mandato DISPLAY APSTATUS.

En algunos casos, el gestor de colas no puede realizar correctamente la conexión a una asociación de instancia de aplicación, en particular:

- Si se han realizado varias conexiones en una conversación compartida desde el mismo proceso, utilizando nombres de aplicación diferentes.
- Si se están utilizando bibliotecas de cliente de un nivel más antiguo. Por ejemplo, las instalaciones de cliente IBM MQ JMS en IBM MQ 9.1.2 y anteriores.

En estas situaciones, si las aplicaciones no se definen a sí mismas como reconectables, esto se permitirá, pero algunas de las agrupaciones de instancias de aplicación podrían ser incorrectas. Si alguna de las conexiones se declara como MQCNO_RECONNECT, esto afecta de forma significativamente negativa al equilibrado de aplicaciones; por lo tanto, la llamada MQCONN se rechazará con MQCNO_RECONNECT_INCOMPATIBLE.

Conceptos relacionados

"Especificación del nombre de aplicación en los lenguajes de programación admitidos" en la página 54
Antes de IBM MQ 9.2.0, ya podría especificar un nombre de aplicación en las aplicaciones cliente de Java o JMS. A partir de IBM MQ 9.2.0, esta característica se amplía a otros lenguajes de programación en IBM MQ for Multiplatforms.

Programas de aplicación que utilizan la MQI

Los programas de aplicación de IBM MQ necesitan varios objetos para poder ejecutarse correctamente.

En la Figura 1 en la página 11, se muestra una aplicación que quita mensajes de una cola, los procesa y luego envía el resultado a otra cola del mismo gestor de colas.

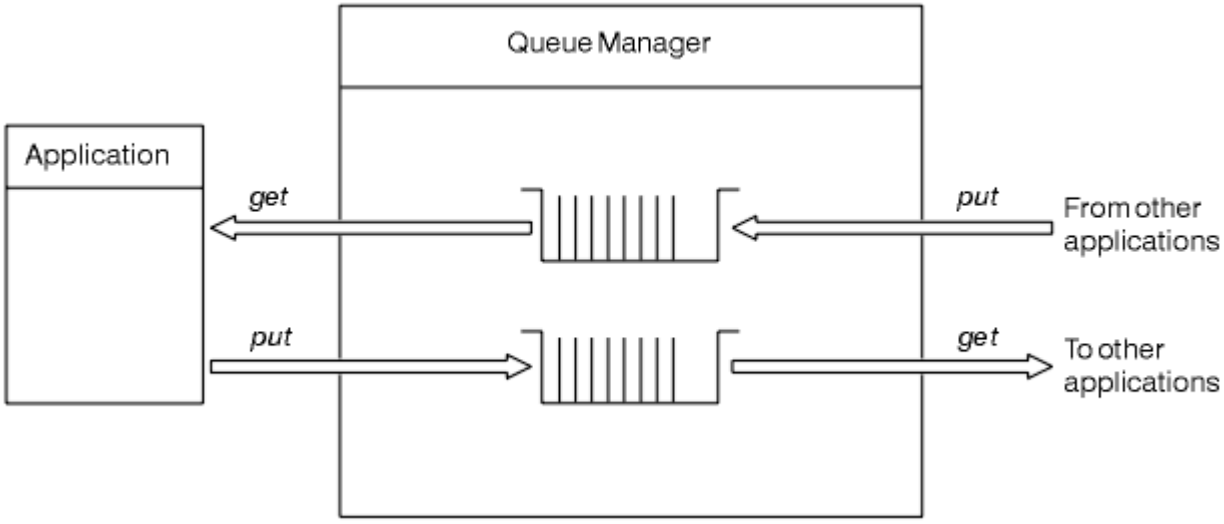


Figura 1. Colas, mensajes y aplicaciones

Aunque las aplicaciones pueden poner mensajes en colas locales o remotas (utilizando MQPUT), solo pueden obtener mensajes directamente de las colas locales (utilizando MQGET).

Para poder ejecutar esta aplicación, deben cumplirse las siguientes condiciones:

- El gestor de colas debe existir y estar en ejecución.
- La primera cola de aplicación, de la que se van a quitar los mensajes, debe estar definida.
- La segunda cola, en la que la aplicación coloca los mensajes, también debe estar definida.
- La aplicación debe poder conectarse con el gestor de colas. Para ello, debe estar enlazada con IBM MQ. Consulte "Creación de una aplicación procedimental" en la página 1025.
- Las aplicaciones que colocan los mensajes en la primera cola deben conectarse también a un gestor de colas. Si son remotas, también deben estar configuradas con colas de transmisión y canales. Esta parte del sistema no aparece en la Figura 1 en la página 11.

Utilización de conexiones cliente para conectarse a varios gestores de colas IBM MQ

Es posible configurar aplicaciones conectadas de cliente para conectarse a más de un gestor de colas (por razones de equilibrio de carga o disponibilidad de servicio).

Los mecanismos principales para conseguirlo en el cliente IBM MQ son el uso de tablas de definición de canal de cliente, consulte Configuración de tablas de definición de canal de clienteo listas de conexiones.

También es posible lograr un comportamiento similar utilizando productos de equilibrio de carga externos o envolviendo el código de conexión IBM MQ en un 'stub ' que puede redireccionar nombres de host o direcciones IP.

Cada una de estas técnicas viene con algunas restricciones, y puede ser más o menos adecuada a los requisitos particulares de la aplicación. Las siguientes secciones, aunque no exhaustivas, describen aspectos particulares que usted debe considerar, y el efecto de estos diferentes enfoques en estos aspectos.

Los clústeres uniformes de IBM MQ , consulte Acerca de los clústeres uniformes, proporcionan un potente mecanismo para lograr el escalado horizontal de aplicaciones en varios gestores de colas basándose en el mecanismo básico de la CCDT para proporcionar varios destinos. Los clústeres uniformes pueden proporcionar prestaciones más allá de lo que es posible utilizando un equilibrador de carga externo que no conoce los protocolos IBM MQ subyacentes, y evitar algunos de los problemas que se describen a continuación, por lo tanto, considere la posibilidad de utilizar un clúster uniforme en lugar de otras técnicas cuando sea aplicable.

Atención: Debe utilizar con precaución las aplicaciones que utilizan IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, incluidas las que utilizan uno de los adaptadores de recursos de IBM MQ , que se conectan a gestores de colas utilizando tecnologías de equilibrio de carga. Si encuentra problemas, vuelva a crearlos sin intentar utilizar el equilibrio de carga.

Hay múltiples temas involucrados, todos los cuales significan que tales conexiones son en el mejor de los casos problemáticas y en el peor totalmente poco confiables:

- Se necesita una atención especial al conectar cualquier aplicación que realice varias conexiones con el gestor de colas utilizando cualquier forma de equilibrio de carga. Esto incluye todas las aplicaciones que utilizan las clases IBM MQ para JMS/Jakarta Messaging ya que crean varias conexiones IBM MQ en uso general. Si se utiliza un equilibrador de carga externo o un apéndice de código personalizado, esto debe direccionar las conexiones de la misma instancia de aplicación al mismo gestor de colas en todo momento.
- El uso de la gestión de transacciones XA o JTA (Java Transaction API) se basa en la capacidad de conectarse de forma coherente al mismo gestor de colas; en la práctica, es poco probable que esto sea práctico con cualquier forma de equilibrio de carga.
- -La gestión uniforme de clústeres se basa en la capacidad de indicar a los clientes que se reconecten a gestores de colas específicos sin interferencias. No es aconsejable intentar combinar el equilibrio de carga externo con el uso de clústeres uniformes

Debe utilizar la funcionalidad de clúster uniforme de IBM MQ para lograr el escalado horizontal de aplicaciones entre varios gestores de colas, en lugar de tecnologías de equilibrio de carga externo. Consulte Configuración de un clúster uniformey los temas siguientes, para obtener información sobre los clústeres uniformes, incluido cómo crear y utilizar clústeres uniformes.

Términos utilizados en esta información

CCDT- multi-QMGR

Significa un archivo CCDT que contiene varios canales de conexión de cliente (CLNTCONN) con el mismo grupo, que es el atributo de conexión de cliente del nombre del gestor de colas (QMNAME CLNTCONN), donde distintas entradas CLNTCONN se resuelven en gestores de colas diferentes.

Esto es distinto de un archivo CCDT que contiene varias entradas CLNTCONN que son simplemente direcciones IP o nombres de host diferentes para el mismo gestor de colas multiinstancia, que es un enfoque que podría elegir para combinar con un apéndice de código.

Si elige un enfoque de gestor de colas múltiple de CCDT, tendrá que elegir si va a priorizar las entradas o si va a tener una gestor de carga de trabajo (WLM) aleatorizado:

Priorizado

Utilice varias entradas ordenadas alfabéticamente con los atributos CLNTWGHT(1) y AFFINITY(PREFERRED) para recordar la última buena conexión.

Aleatorizado

Utilice los atributos CLNTWGHT(1) y AFFINITY(NONE). Puede ajustar la ponderación de WLM entre servidores IBM MQ escalados de forma diferente ajustando el valor CLNTWGHT

Nota: Debería evitar grandes diferencias en CLNTWGHT entre los canales.

Equilibrador de carga

Significa un dispositivo de red con una dirección de IP virtual (VIP) configurada con la supervisión de puerto de los escuchas TCP/IP de varios gestores de colas IBM MQ. La forma en la que se configura la VIP en el dispositivo de red depende del dispositivo de red que está utilizando.

Las opciones siguientes solo están relacionadas con las aplicaciones que envían mensajes o inician la mensajería síncrona de solicitud y respuesta. Las consideraciones para las aplicaciones que dan servicio a estos mensajes y solicitudes, por ejemplo, los escuchas están completamente separados y se describen en detalle en "Conexión de un escucha de mensajes a una cola".

Escala de cambio de código necesario para las aplicaciones existentes que se conectan a un único gestor de colas

lista CONNAME, CCDT multi-QMGR y equilibrador de carga
MQCONN("QMNAME") en MQCONN("*QMNAME")

El nombre del gestor de colas puede estar en la configuración JNDI (Java Naming and Directory Interface) para aplicaciones Java Platform, Enterprise Edition (Java EE). De lo contrario, esto requiere un cambio de código de caracteres.

Apéndice de código
Sustituya la lógica de conexión JMS o MQI existentes con un apéndice de código.

Soporte para estrategias de WLM diferentes

Lista CONNAME
Solo priorizado.

Probablemente, esto va a tener un efecto negativo en el código.

CCDT multi-QMGR
Priorizado o aleatorio.

Probablemente, esto no va a tener ningún efecto sobre el código.

Equilibrador de carga
Cualquiera, incluyendo cada conexión para todos los mensajes.

Probablemente, esto va a tener un efecto positivo en el código.

Apéndice de código
Cualquiera, incluyendo cada mensaje para todos los mensajes.

Probablemente, esto va a tener un efecto positivo en el código.

Sobrecarga de rendimiento mientras el gestor de colas principal no está disponible

Lista CONNAME
Siempre intenta el primero de la lista.

Probablemente, esto va a tener un efecto negativo en el código.

CCDT multi-QMGR
Recuerda la última buena conexión.

Probablemente, esto va a tener un efecto positivo en el código.

Equilibrador de carga
La supervisión de puerto evita gestores de colas incorrectos.

Probablemente, esto va a tener un efecto positivo en el código.

Apéndice de código
Puede recordar la última buena conexión y reintentar de forma inteligente.

Probablemente, esto va a tener un efecto positivo en el código.

Soporte de transacciones XA

lista CONNAME, CCDT multi-QMGR y equilibrador de carga
El gestor de transacción tiene que almacenar la información de recuperación que se reconecta al recurso del mismo gestor de colas.

Por regla general, una llamada MQCONN que se resuelve en gestores de colas diferentes lo invalida. Por ejemplo, en Java EE, una única fábrica de conexiones se debe resolver en un único gestor de colas cuando se utiliza XA.

Probablemente, esto va a tener un efecto negativo en el código.

Apéndice de código
El apéndice de código puede cumplir los requisitos de XA para un gestor de transacción, por ejemplo, varias fábricas de conexiones.

Probablemente, esto va a tener un efecto positivo en el código.

Flexibilidad de administración para ocultar los cambios de infraestructura de las apps

Lista CONNAME
Solo DNS.

Probablemente, esto va a tener un efecto negativo en el código.

CCDT multi-QMGR
DNS y sistemas de archivos compartidos, o sistema de archivos compartidos, o envío (push) de archivo de CCDT.

Probablemente, esto no va a tener ningún efecto sobre el código.

Equilibrador de carga
Dirección IP virtual (VIP) dinámica.

Probablemente, esto va a tener un efecto positivo en el código.

Apéndice de código
Entradas CCDT de DNS o de un único gestor de colas.

Probablemente, esto no va a tener ningún efecto sobre el código.

Cómo evitar interrupciones en el mantenimiento planificado

Existe otra situación que debe tener en cuenta y para la que debe realizar una planificación, que es cómo evitar las interrupciones en las aplicaciones, por ejemplo, errores y tiempos de espera excedidos visibles para los usuarios finales, durante el mantenimiento planificado de un gestor de colas. El mejor enfoque para evitar las interrupciones es eliminar todo el trabajo de un gestor de colas antes de que se detenga.

Considere un escenario de solicitud y respuesta. Desea que se completen todas las solicitudes en curso y las respuestas que van a ser procesadas por la aplicación, pero no desea que se envíe ningún trabajo adicional en el sistema. Simplemente la inmovilización del gestor de colas no satisface esta necesidad, ya que las aplicaciones bien codificadas reciben una excepción MQRC_Q_MGR_QUIESCING con código de retorno RC2161, antes de que reciban sus mensajes de respuesta para las solicitudes en curso.

Puede establecer PUT(DISABLED) en las colas de solicitud utilizadas para enviar trabajo, mientras deja las colas de respuesta como PUT(ENABLED) y también GET(ENABLED). De esta forma, puede supervisar la profundidad de las colas de solicitud, transmisión y de respuesta. Una vez que todas se hayan estabilizado, es decir, que las solicitudes en curso se completen o agoten su tiempo de espera, puede detener el gestor de colas.

Sin embargo, es necesaria una buena codificación en las aplicaciones solicitantes para manejar una cola de solicitud PUT(DISABLED), que genera el error MQRC_PUT_INHIBITED con el código de error RC2051, cuando se intenta enviar un mensaje.

Tenga en cuenta que la excepción no se produce cuando se crea la conexión con IBM MQ, ni al abrir la cola de solicitud. La excepción solo se produce cuando se realiza un intento para enviar realmente un mensaje, utilizando la llamada MQPUT.

La creación de un apéndice de código que incluya esta lógica de manejo de errores para los escenarios de solicitud y respuesta y pedir a los equipos de aplicaciones que utilicen dicho apéndice de código en el futuro pueden ayudarle a desarrollar aplicaciones con un comportamiento coherente.

V 9.3.0 Desarrollo de aplicaciones cliente flexibles y escalables

Para la tolerancia a errores y la escalabilidad, el despliegue de aplicaciones cliente que dan soporte a las opciones de conexiones en clústeres uniformes permite que las instancias de la aplicación se reequilibren entre gestores de colas.

Consulte Acerca de los clústeres uniformes para obtener una visión general de los clústeres uniformes.

Idealmente, este reequilibrio es invisible para la aplicación, pero sólo determinados tipos de aplicación son adecuados para este tipo de despliegue, y es posible que sea necesario tener en cuenta alguna consideración en el diseño de la aplicación.

Estas consideraciones se dividen en dos categorías principales:

- *vías de acceso de error* raras que pueden existir ya para aplicaciones reconectables, pero que son más probables cuando se despliegan en un clúster uniforme. Por ejemplo, después de una reconexión, se restituye cualquier unidad de trabajo en curso y se restablecen los cursores de examen. Esto puede ser un suceso raro para la aplicación reconectable en su entorno actual y, por lo tanto, no se maneja de la forma más limpia posible mediante el código de aplicación. La revisión de la lógica de la aplicación, para asegurarse de que existe un manejo adecuado para estas situaciones, ayuda a evitar que surjan problemas inesperados.
- *Afinidades* a un gestor de colas determinado. Si sabe que una aplicación siempre debe volver a conectarse al mismo gestor de colas o a un gestor de colas específico, la aplicación debe estar configurada para volver a conectarse a ese gestor de colas, o no tener habilitada su conexión con ese gestor de colas. Sin embargo, estas afinidades pueden ser temporales, como esperar un mensaje de respuesta. En la sección siguiente se describe cómo influir en el algoritmo de equilibrio para tener en cuenta estas afinidades del código de aplicación. Para obtener más detalles sobre estas opciones y cómo lograr un enfoque similar a través de la configuración, en lugar del código de aplicación, consulte Influenciar el reequilibrio de aplicaciones en clústeres uniformes.

Influencia en las opciones de reconexión en la MQI

Consulte Opciones de reconexión para obtener más información sobre MQCNO_RECONNECT.

Si sabe que una aplicación siempre debe volver a conectarse al mismo gestor de colas o a un gestor de colas específico, debe configurarse como MQCNO_RECONNECT_Q_MGR o MQCNO_RECONNECT_DISABLED.

Influencia en el algoritmo de equilibrio en la MQI

Sin embargo, es posible que desee controlar o influir en ese comportamiento de reequilibrio para que se ajuste a las necesidades de tipos específicos de aplicación; por ejemplo, minimizar las interrupciones en las transacciones en curso o asegurarse de que las aplicaciones solicitantes reciben sus respuestas antes de que se muevan.

Se asumen y se describen determinados comportamientos deseables predeterminados en Influenciar el reequilibrio de aplicaciones en clústeres uniformes. También puede influir en el comportamiento de aplicaciones específicas durante la configuración o el despliegue a través del archivo client.ini tal como se describe en dicho tema.

En otras situaciones, puede tener más sentido hacer que el comportamiento de equilibrio y los requisitos formen parte de la lógica de la aplicación. En estos casos, se pueden proporcionar las mismas características relevantes de la aplicación a IBM MQ al conectarse al gestor de colas en la llamada MQCONNX, en una estructura denominada MQBNO (opciones de equilibrio).

Si proporciona una estructura MQBNO, debe proporcionar toda la información que necesita IBM MQ para tomar una decisión sobre cómo y cuándo se debe solicitar a la aplicación que se vuelva a conectar a un gestor de colas diferente.

Debe proporcionar:

- El **Type** de la aplicación

- El **Timeout** en el que se reequilibra la instancia independientemente del estado
- Cualquier **BalanceOptions** especial

La excepción a esto es que puede dejar el tiempo de espera como MQBNO_TIMEOUT_DEFAULT si es necesario. En este caso, el tiempo de espera se resuelve en cualquier valor del archivo client.ini , la aplicación o las stanzas globales, si se proporcionan, y en su defecto, en el valor predeterminado base de 10 segundos.

Consulte MQBNO para obtener detalles sobre el formato de esta estructura.

Para aplicaciones .NET, consulte Influenciar el reequilibrio de aplicaciones en .NET para obtener más información.

Aplicaciones orientadas a objetos

IBM MQ proporciona soporte para JMS, Java, C++ y .NET. Estos lenguajes e infraestructuras utilizan el mismo modelo de objeto de IBM MQ que proporciona clases que incluyen las mismas funciones que las llamadas y estructuras de IBM MQ.

Algunos de los lenguajes e infraestructuras que utilizan el modelo de objetos de IBM MQ proporcionan funciones adicionales que no están disponibles cuando utiliza los lenguajes de procedimiento con la interfaz de colas de mensajes (MQI).

Para obtener información detallada acerca de las clases, métodos y propiedades que proporciona este modelo, consulte la sección “Modelo de objeto IBM MQ” en la página 17.

JMS

IBM MQ proporciona clases que implementan las especificaciones Jakarta Messaging 3.0 y Java Message Service 2.0 . Para obtener detalles de IBM MQ classes for JMS, consulte Utilización de IBM MQ classes for JMS. Para obtener información sobre las diferencias entre IBM MQ classes for Java y IBM MQ classes for JMS, para ayudarle a decidir qué utilizar, consulte “Desarrollo de aplicaciones JMS/Jakarta Messaging y Java” en la página 84.

IBM MQ Message Service Client (XMS) for C/C++ y IBM MQ Message Service Client (XMS) for .NET proporciona una interfaz de programación de aplicaciones (API), denominada XMS que tiene el mismo conjunto de interfaces que la API de Java Message Service (JMS). Para obtener más información, consulte “Desarrollo de aplicaciones XMS .NET” en la página 633.

Java

Consulte la sección Utilización de IBM MQ classes for Java para obtener información acerca de cómo codificar programas utilizando el modelo de objetos IBM MQ en Java.

Stabilized IBM no hará mejoras adicionales en IBM MQ classes for Java y sus funciones se estabilizarán en el nivel suministrado en IBM MQ 8.0. Para obtener información acerca de las diferencias entre las IBM MQ classes for Java y las IBM MQ classes for JMS y cómo decidir cuál de ellas ha de utilizar, consulte la sección “Desarrollo de aplicaciones JMS/Jakarta Messaging y Java” en la página 84.

C++

IBM MQ proporciona clases C++ equivalentes a los objetos IBM MQ y algunas clases equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI. Consulte Utilización de C++ para obtener información sobre la codificación de programas utilizando IBM MQ Object Model en C++. Message Service Clients for C/C++ y .NET proporcionan una interfaz de programación de aplicaciones (API) denominada XMS que tiene el mismo conjunto de interfaces que Java Message Service (JMS) API.

.NET

Consulte Desarrollo de aplicaciones .NET para obtener información acerca de cómo codificar programas .NET utilizando las clases de IBM MQ .NET. Message Service Clients para C/C++ y .NET proporciona una interfaz de programación de aplicaciones (API), con el nombre XMS, que tiene el mismo conjunto de interfaces que la API de Java Message Service (JMS).

Conceptos relacionados

"Desarrollo de aplicaciones MQI con IBM MQ" en la página 736
IBM MQ proporciona soporte para C, Visual Basic, COBOL, Assembler, RPG, pTALy PL/I. Estos lenguajes de procedimiento utilizan la interfaz de cola de mensajes (MQI) para acceder a los servicios de colas de mensajes.

Visión general técnica

"Conceptos de desarrollo de aplicaciones" en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

Referencia relacionada

Guía de consulta para el desarrollo de aplicaciones

Modelo de objeto IBM MQ

El modelo de objeto de IBM MQ consta de clases, métodos y propiedades.

El modelo de objeto de IBM MQ consta de:

- *Clases*, que representan conceptos conocidos de IBM MQ como, por ejemplo, los gestores de colas, las colas y los mensajes.
- *Métodos* en cada clase correspondiente a las llamadas MQI.
- *Propiedades* en cada clase correspondiente a los atributos de los objetos de IBM MQ.

Cuando se crea una aplicación de IBM MQ utilizando el modelo de objeto de IBM MQ, se crean instancias de estas clases en la aplicación. Una instancia de una clase en la programación orientada a objetos se denomina un *objeto*. Una vez creado un objeto, se interactúa con el objeto examinando o estableciendo los valores de las propiedades del objeto (el equivalente de emitir una llamada MQINQ o MQSET), y realizando llamadas de método en el objeto (el equivalente de emitir las otras llamadas MQI).

Clases

El modelo de objeto de IBM MQ proporciona el siguiente conjunto base de clases.

La implementación real del modelo varía ligeramente entre los distintos entornos orientados a objetos soportados.

MQQueueManager
Un objeto de la clase MQQueueManager representa una conexión con un gestor de colas. Tiene métodos Connect(), Disconnect(), Commit() y Backout() (el equivalente de MQCONN o MQCONNX, MQDISC, MQCMIT y MQBACK). Tiene propiedades correspondientes a los atributos de un gestor de colas. El acceso a una propiedad de atributo de gestor de colas se conecta implícitamente al gestor de colas si no está ya conectado. La destrucción de un objeto MQQueueManager se desconecta implícitamente del gestor de colas.

MQQueue
Un objeto de la clase MQQueue representa una cola. Tiene métodos Put() y Get() para poner y obtener mensajes en la cola (el equivalente de MQPUT y MQGET). Tiene propiedades correspondientes a los atributos de una cola. El acceso a una propiedad de atributo de cola o la emisión de una llamada de método Put() o Get() abre implícitamente la cola (el equivalente de MQOPEN). La destrucción de un objeto MQQueue cierra implícitamente la cola (el equivalente de MQCLOSE).

MQTopic
Un objeto de la clase MQTopic representa un tema. Tiene métodos Put() (publicar) y Get() (recibir o suscribir) para poner y obtener mensajes en el tema (el equivalente de MQPUT y MQGET). Tiene propiedades correspondientes a los atributos de un tema. Solo puede accederse a un objeto MQTopic para la publicación o la suscripción, no para ambos simultáneamente. Cuando se utiliza para recibir mensajes, el objeto MQTopic puede crearse con una suscripción gestionada o no gestionada, y como un suscriptor duradero o no duradero; se proporcionan varios constructores sobrecargados para los distintos escenarios.

MQMessage
Un objeto de la clase MQMessage representa un mensaje que se va a poner en una cola u obtener de una cola. Contiene un almacenamiento intermedio y encapsula los datos de aplicación y MQMD. Tiene propiedades correspondientes a campos MQMD y métodos que le permiten escribir y leer datos de usuario de distintos tipos (por ejemplo, series, enteros largos, enteros cortos, bytes individuales) hacia y desde el almacenamiento intermedio.

MQPutMessageOptions
Un objeto de la clase MQPutMessageOptions representa la estructura MQPMO. Tiene propiedades correspondientes a los campos MQPMO.

MQGetMessageOptions
Un objeto de la clase MQGetMessageOptions representa la estructura MQGMO. Tiene propiedades correspondientes a los campos MQGMO.

MQProcess
Un objeto de la clase MQProcess representa una definición de proceso (que se utiliza con el desencadenante). Tiene propiedades que representan los atributos de una definición de proceso.

Multi **MQDistributionList**
Un objeto de la clase MQDistributionList representa una lista de distribución (que se utiliza para enviar varios mensajes con una sola MQPUT). Contiene una lista de objetos MQDistributionListItem.

Multi **MQDistributionListItem**
Un objeto de la clase MQDistributionListItem representa un único destino de lista de distribución. Encapsula las estructuras MQOR, MQRR y MQPMR, y tiene propiedades correspondientes a los campos de estas estructuras.

Referencias de objetos

En un programa de IBM MQ que utiliza MQI, IBM MQ devuelve los manejadores de conexión y los manejadores de objetos al programa.

Estos manejadores deben pasarse como parámetros en las llamadas de IBM MQ posteriores. Con el modelo de objeto de IBM MQ, estos manejadores se ocultan del programa de aplicación. En su lugar, la creación de un objeto de una clase da como resultado la devolución de una referencia de objeto al programa de aplicación. Esta es la referencia de objeto que se utiliza cuando se realizan llamadas de método y accesos de propiedades en el objeto.

Códigos de retorno

La emisión de una llamada de método o el establecimiento de un valor de propiedad da como resultado el establecimiento de códigos de retorno.

Estos códigos de retorno son un código de terminación y un código de razón, y son a su vez propiedades del objeto. Los valores de código de terminación y código de razón son los mismos que los definidos para MQI, con algunos valores adicionales específicos del entorno orientado a objetos.

Mensajes de IBM MQ

Un mensaje de IBM MQ está formado por propiedades del mensaje y datos de aplicación. El descriptor de mensaje de colocación de mensajes en colas (MQMD) contiene la información de control que acompaña a los datos de la aplicación cuando un mensaje viaja entre las aplicaciones de envío y recepción.

Partes de un mensaje

Los mensajes de IBM MQ constan de dos partes:

- Propiedades del mensaje
- Datos de la aplicación

La Figura 2 en la página 19 representa un mensaje y muestra cómo se divide lógicamente en las propiedades de los mensajes y los datos de aplicación.

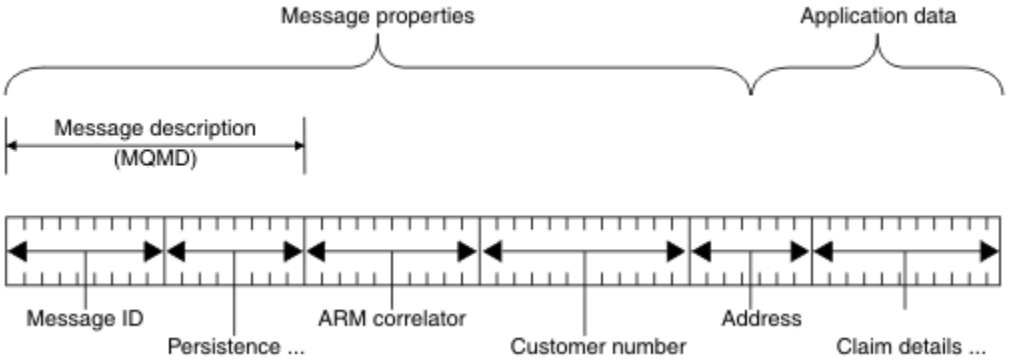


Figura 2. Representación de un mensaje

Los datos de aplicación que se transportan en un mensaje de IBM MQ no los puede cambiar el gestor de colas a menos que se lleve a cabo la conversión de datos en él. Además, IBM MQ no pone ninguna restricción en el contenido de estos datos. La longitud de los datos en cada mensaje no puede exceder el valor del atributo **MaxMsgLength** de la cola y el gestor de colas.

ALW En AIX, Linux, and Windows, el atributo *MaxMsgLength* del gestor de colas y la cola toma como valor predeterminado 4 MB (4 194 304 bytes) que puede cambiar hasta un máximo de 100 MB (104 857 600 bytes) si es necesario.

IBM i En IBM i, el atributo *MaxMsgLength* del gestor de colas y la cola toma como valor predeterminado 4 MB (4 194 304 bytes) que puede cambiar hasta un máximo de 100 MB (104 857 600 bytes) si es necesario. Si tiene previsto utilizar mensajes de IBM MQ de más de 15 MB en IBM i, consulte “Creación de una aplicación procedimental en IBM i” en la página 1032.

z/OS En z/OS, el atributo **MaxMsgLength** del gestor de colas se fija en 100 MB y el atributo **MaxMsgLength** de la cola toma como valor predeterminado 4 MB (4 194 304 bytes), que puede cambiar hasta un máximo de 100 MB si es necesario.

Acorte un poco los mensajes a una longitud menor que el valor del atributo **MaxMsgLength** en algunos casos. Para obtener más información, consulte “Los datos del mensaje” en la página 777.

Puede crear un mensaje cuando utiliza las llamadas MQI MQPUT o MQPUT1. Como entrada para estas llamadas, puede proporcionar la información de control (como la prioridad del mensaje y el nombre de una cola de respuestas) y los datos y, a continuación, la llamada coloca el mensaje en una cola. Consulte MQPUT y MQPUT1 para obtener más información sobre estas llamadas.

Descriptor de mensaje

Puede acceder a la información de control de mensajes utilizando la estructura MQMD, que define el *descriptor de mensaje*.

Para obtener una descripción completa de la estructura MQMD, consulte MQMD - Descriptor de mensaje.

Para obtener una descripción de cómo utilizar los campos dentro del MQMD que contienen información sobre el origen del mensaje, consulte “Contexto de mensaje” en la página 48.

Hay distintas versiones del descriptor de mensaje. En la versión 2 del descriptor de mensaje (o MQMDE), encontrará información adicional para agrupar y segmentar mensajes (consulte “Grupos de mensajes” en la página 45). Es igual que el descriptor de mensaje de la Versión 1, pero tiene campos adicionales. Estos campos se describen en MQMDE - Extensión de descriptor de mensaje.

Tipos de mensajes

Hay cuatro tipos de mensajes definidos por IBM MQ.

Los cuatro mensajes son:

- Datagrama
- Mensajes de solicitud
- Mensajes de respuesta
- Mensajes de informe
  - Tipos de mensaje de informe
  - Opciones de mensaje de informe

Las aplicaciones pueden utilizar los tres primeros tipos de mensajes para pasar información entre ellos. El cuarto tipo, informe, es para que lo utilicen las aplicaciones y los gestores de colas para informar sobre sucesos como la aparición de un error.

Cada tipo de mensaje se identifica mediante un valor MQMT_*. También puede definir sus propios tipos de mensajes. Para conocer el rango de valores que puede utilizar, consulte MsgType.

Datagramas

Utilice un *datagrama* cuando no necesita una respuesta de la aplicación que recibe el mensaje (es decir, obtiene el mensaje de la cola).

Un ejemplo de una aplicación que puede utilizar datagramas es la que muestra información de vuelos en la sala de un aeropuerto. Un mensaje puede contener los datos de una pantalla completa de información de vuelos. Este tipo de aplicación es poco probable que solicite un acuse de recibo de un mensaje, porque probablemente no importa si el mensaje no se entrega. La aplicación envía un mensaje de actualización después de un breve periodo de tiempo.

Mensajes de solicitud

Utilice un *mensaje de solicitud* cuando desee una respuesta de la aplicación que recibe el mensaje.

Un ejemplo de una aplicación que puede utilizar mensajes de solicitud es la que muestra el saldo de una cuenta corriente. El mensaje de solicitud puede contener el número de cuenta y el mensaje de respuesta puede contener el saldo de la cuenta.

Si desea enlazar el mensaje de respuesta con el mensaje de solicitud, hay dos opciones:

- Asegurarse de que la aplicación que maneja el mensaje de solicitud es responsable de garantizar que transfiere información al mensaje de respuesta que se relaciona con el mensaje de solicitud.
- Utilizar el campo de informe en el descriptor de mensaje del mensaje de solicitud para especificar el contenido de los campos *MsgId* y *CorrelId* del mensaje de respuesta:
  - Puede solicitar que el campo *MsgId* o *CorrelId* del mensaje original se copie en el campo *CorrelId* del mensaje de respuesta (la acción predeterminada es copiar *MsgId*).
  - Puede solicitar que se genere un nuevo *MsgId* para el mensaje de respuesta, o que el campo *MsgId* del mensaje original se copie en el campo *MsgId* del mensaje de respuesta (la acción predeterminada es generar un nuevo identificador de mensaje).

Mensajes de respuesta

Utilice un *mensaje de respuesta* cuando responda a otro mensaje.

Cuando cree un mensaje de respuesta, respete las opciones que se han establecido en el descriptor de mensaje del mensaje al que está respondiendo. Las opciones de informe especifican el contenido de los campos de identificador de mensaje (*MsgId*) y de identificador de correlación (*CorrelId*). Estos campos permiten que la aplicación que recibe la respuesta correlacione la respuesta con la solicitud original.

Mensajes de informe

Los *mensajes de informe* informan a las aplicaciones sobre sucesos como la aparición de un error al procesar un mensaje.

Los puede generar:

- Un gestor de colas,
- Un agente de canal de mensajes (por ejemplo, si no pueden entregar el mensaje) o bien
- Una aplicación (por ejemplo, si no puede utilizar los datos del mensaje).

Los mensajes de informe se pueden generar en cualquier momento y pueden llegar a una cola cuando la aplicación no los esperaba.

Tipos de mensaje de informe

Al poner un mensaje en una cola, puede seleccionar que se reciba:

- Un *mensaje de informe de excepción*. Se envía como respuesta a un mensaje con un conjunto de distintivos de excepciones. Lo genera el agente de canal de mensaje (MCA) o la aplicación.
- Un *mensaje de informe de caducidad*. Indica que una aplicación ha intentado recuperar un mensaje que había alcanzado el umbral de caducidad; el mensaje se marca para descartarse. Este tipo de informe lo genera el gestor de colas.
- Un *mensaje de informe de confirmación de llegada (COA)* Indica que el mensaje ha llegado a su cola de destino. Lo genera el gestor de colas.
- Un *mensaje de informe de confirmación de entrega (COD)*. Indica que el mensaje lo ha recuperado una aplicación receptora. Lo genera el gestor de colas.
- Un *mensaje de informe de notificación de acción positiva (PAN)*. Indica que una solicitud se ha atendido satisfactoriamente (es decir, la acción solicitada en el mensaje se ha realizado satisfactoriamente). Este tipo de informe lo genera la aplicación.
- Un *mensaje de informe de notificación de acción negativa (NAN)*. Indica que una solicitud no se ha atendido satisfactoriamente (es decir, la acción solicitada en el mensaje no se ha realizado satisfactoriamente). Este tipo de informe lo genera la aplicación.

Nota: Cada tipo de mensaje de informe contiene uno de los elementos siguientes:

- El mensaje original entero
- Los 100 primeros bytes de datos del mensaje original
- Ningún dato del mensaje original

Puede solicitar más de un tipo de mensaje de informe cuando pone un mensaje en una cola. Cuando selecciona las opciones del mensaje de informe de confirmación de entrega y del mensaje de informe de excepción, si el mensaje no consigue entregarse, recibe un mensaje de informe de excepción. Sin embargo, si solo selecciona la opción del mensaje de informe de confirmación de entrega y el mensaje no consigue entregarse, no obtiene un mensaje de informe de excepción.

Los mensajes de informe que solicita, cuando los criterios para generar un mensaje concreto se cumplen, son los únicos que recibe.

Opciones de mensaje de informe

Puede *descartar* un mensaje después de que haya surgido una excepción. Si selecciona la opción de descartar y ha solicitado un mensaje de informe de excepción, el mensaje de informe va a *ReplyToQ* y *ReplyToQMgr*, y el mensaje original se descarta.

Nota: Una de las ventajas es que puede reducir el número de mensajes que van a la cola de mensajes no entregados. Pero también significa que la aplicación, a menos que solo envíe mensajes de datagrama, tiene que manejar los mensajes devueltos. Cuando se genera un mensaje de informe de excepción, hereda la persistencia del mensaje original.

Si un mensaje de informe no se puede entregar (si la cola está llena, por ejemplo), el mensaje de informe se coloca en la cola de mensajes no entregados.

Si desea recibir un mensaje de informe, especifique el nombre de la cola de respuestas en el campo *ReplyToQ*; de lo contrario, la llamada MQPUT o MQPUT1 del mensaje original falla con MQRC_MISSING_REPLY_TO_Q.

Puede utilizar otras opciones de informe en el descriptor de mensaje (MQMD) de un mensaje para especificar el contenido de los campos *MsgId* y *CorrelId* de cualquier mensaje de informe que se cree para el mensaje:

- Puede solicitar que los campos *MsgId* o *CorrelId* del mensaje original se copien en el campo *CorrelId* del mensaje de informe. La acción predeterminada es copiar el identificador de mensaje. Utilice MQRO_COPY_MSG_ID_TO_CORRELID porque permite que el remitente de un mensaje correlacione el mensaje de respuesta o de informe con el mensaje original. El identificador de correlación del mensaje de respuesta o de informe es idéntico al identificador de mensaje del mensaje original.
- Puede solicitar que se genere un nuevo *MsgId* para el mensaje de informe o bien que el campo *MsgId* del mensaje original se copie en el campo *MsgId* del mensaje de informe. La acción predeterminada es generar un nuevo identificador de mensaje. Utilice MQRO_NEW_MSG_ID, ya que garantiza que cada mensaje del sistema tenga un identificador de mensaje diferente y que se pueda distinguir claramente de los demás mensajes del sistema.
- Las aplicaciones especializadas podrían necesitar utilizar MQRO_PASS_MSG_ID o MQRO_PASS_CORREL_ID. No obstante, debe diseñar una aplicación que lea los mensajes de la cola para asegurarse de que funciona correctamente cuando, por ejemplo, la cola contiene varios mensajes con el mismo identificador de mensaje.

  Las aplicaciones de servidor deben comprobar los valores de estos distintivos en el mensaje de solicitud y establecer correctamente los campos *MsgId* y *CorrelId* en el mensaje de respuesta o de informe.

  Las aplicaciones que actúan como intermediarios entre una aplicación solicitante y una aplicación de servidor no necesitan comprobar los valores de estos distintivos. Esto se debe a que normalmente estas aplicaciones tienen que reenviar el mensaje a la aplicación de servidor con los campos *MsgId*, *CorrelId* y *Report* sin modificar. Esto permite a la aplicación de servidor copiar *MsgId* del mensaje original en el campo *CorrelId* del mensaje de respuesta.

Al generar un informe sobre un mensaje, las aplicaciones de servidor deben comprobar si se ha establecido alguna de estas opciones.

Para obtener más información sobre cómo utilizar los mensajes de informe, consulte Report.

Para indicar la naturaleza del informe, los gestores de colas utilizan un rango de códigos de feedback. Colocan estos códigos en el campo *Feedback* del descriptor de mensaje de un mensaje de informe. Los gestores de colas también pueden devolver códigos de razón MQI en el campo *Feedback*. IBM MQ define el rango de códigos de feedback que utilizan las aplicaciones.

Para obtener más información sobre los códigos de razón y de feedback, consulte Feedback.

Un ejemplo de un programa que puede utilizar un código de feedback es el que supervisa las cargas de trabajo de otros programas que atienden una cola. Si hay más de una instancia de un programa que sirve a una cola y el número de mensajes que llegan a la cola ya no lo justifica, un programa de este tipo puede enviar un mensaje de informe (con el código de feedback MQFB_QUIT) a uno de los programas de servicio para indicar que el programa debe terminar su actividad. (Un programa de supervisión puede utilizar la llamada MQINQ para averiguar cuántos programas están dando servicio a una cola).

Multi *Informes y mensajes segmentados*

No está soportado en IBM MQ for z/OS.

Si un mensaje está segmentado y solicita que se generen informes, es posible que reciba más informes que los que hubiera recibido si el mensaje no estuviera segmentado.

Para ver una descripción de los mensajes segmentados, consulte “Segmentación de mensajes” en la página 813.

Para informes generados por IBM MQ

Si segmenta los mensajes o permite que el gestor de colas lo haga, solo hay un caso en el que podría esperar recibir un único informe para el mensaje completo. Este sería cuando ha solicitado solo informes COD y ha especificado MQGMO_COMPLETE_MSG en la aplicación de obtención.

En otros casos, su aplicación debe estar preparada para hacer frente a varios informes, normalmente uno por cada segmento.

Nota: Si segmenta los mensajes y solo necesita que se devuelvan los primeros 100 bytes de datos del mensaje original, cambie el valor de las opciones de informe para solicitar informes sin datos para segmentos que tengan un desplazamiento de 100 o más. Si no lo hace y deja el valor de forma que cada segmento solicite 100 bytes de datos, y recupera los mensajes de informe con un MQGET individual especificando MQGMO_COMPLETE_MSG, los informes se agrupan en un mensaje grande que contiene 100 bytes de datos leídos en cada desplazamiento adecuado. Si esto ocurre, necesitará un almacenamiento intermedio grande o deberá especificar MQGMO_ACCEPT_TRUNCATED_MSG.

Para informes generados por aplicaciones

Si su aplicación genera informes, copie siempre las cabeceras de IBM MQ que estén presentes al principio de los datos del mensaje original en los datos del mensaje del informe.

A continuación, añada ninguno, 100 bytes o todos los datos del mensaje original (o cualquier otra cantidad que incluya normalmente) a los datos de mensaje de informe.

Puede reconocer las cabeceras de IBM MQ que se deben copiar mirando los nombres de formato (Format)sucesivos, comenzando por MQMD y siguiendo por las cabeceras presentes. Los nombres de `Format` siguientes indican estas cabeceras de IBM MQ:

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* significa cualquier nombre que comience por los caracteres MQH.

El nombre `Format` aparece en posiciones específicas para MQDLH y MQXQH, pero para las demás cabeceras de IBM MQ aparece en la misma posición. La longitud de la cabecera está incluida en un campo que también aparece en la misma posición para MQMDE, MQIMS y todas las cabeceras MQH*.

Si utiliza una MQMD Versión 1 y está creando informes sobre un segmento, un mensaje en un grupo o un mensaje para el que se permite la segmentación, los datos de informe deben comenzar por una MQMDE. Establezca el campo *`OriginalLength`* en la longitud de los datos del mensaje original excluyendo las longitudes de las cabeceras IBM MQ que encuentre.

Recuperación de informes

Si solicita informes COA o COD, puede solicitar que se reagrupen con MQGMO_COMPLETE_MSG.

Una solicitud MQGET con MQGMO_COMPLETE_MSG se satisface cuando hay suficientes mensajes de informe (de un tipo individual, por ejemplo COA, y con el mismo *`GroupId`*) presentes en la cola para representar un mensaje original completo. Esto es cierto incluso aunque los propios mensajes de informe no contienen los datos originales completos; el campo *`OriginalLength`* de cada mensaje de informe proporciona la longitud de los datos originales representados por dicho mensaje de informe, incluso si los propios datos no están presentes.

Puede utilizar esta técnica incluso si hay varios tipos de informes distintos presentes en la cola (por ejemplo, tanto COA como COD), debido a que MQGET con MQGMO_COMPLETE_MSG reagrupa mensajes de informe únicamente si tienen el mismo código *Feedback*. No obstante, normalmente no se puede utilizar esta técnica para informes de excepción, ya que, en general, estos tienen códigos *Feedback* distintos.

Puede utilizar esta técnica para obtener una indicación positiva de que ha llegado el mensaje completo. Sin embargo, en la mayoría de las circunstancias, necesitará contemplar la posibilidad de que lleguen algunos segmentos mientras que otros generen una excepción (o caduquen, si está permitido). No puede utilizar MQGMO_COMPLETE_MSG en este caso porque, en general, podría obtener códigos *Feedback* diferentes para distintos segmentos y podría obtener más de un informe para un segmento. No obstante, puede utilizar MQGMO_ALL_SEGMENTS_AVAILABLE.

Para permitir esto, es posible que necesite recuperar informes a medida que llegan, y crear una imagen en su aplicación de lo que ha ocurrido con el mensaje original. Puede utilizar el campo *GroupId* del mensaje de informe para correlacionar informes con el *GroupId* del mensaje original, y el campo *Feedback* para identificar el tipo de cada mensaje de informe. La forme en que se hace esto dependerá de los requisitos de su aplicación.

Un enfoque es el siguiente:

- Solicite informes COD e informes de excepción.
- Después de un periodo de tiempo específico, compruebe si se ha recibido un conjunto completo de informes COD utilizando MQGMO_COMPLETE_MSG. Si es así, su aplicación sabe que se ha procesado el mensaje completo.
- Si no es así, y hay informes de excepción presentes en relación a este mensaje, gestione el problema como lo haría para mensajes no segmentados , pero asegúrese de limpiar los segmentos huérfanos en algún momento.
- Si hay segmentos para los cuales no hay informes de ningún tipo, es posible que los segmentos originales (o los informes) estén esperando a que se reconecte un canal, o que la red se pueda haber sobrecargado en algún momento. Si no se ha recibido ningún informe de excepción (o si piensa que los que tiene pueden ser solo temporales), puede optar por dejar que su aplicación espere un poco más.

  Como antes, esto es similar a las consideraciones que debe plantearse al tratar con mensajes no segmentados, excepto por el hecho de que también debe tener en cuenta la posibilidad de limpiar segmentos huérfanos.

Si el mensaje original no es crítico (por ejemplo, si es una consulta o un mensaje que se puede repetir más adelante), establezca un tiempo de caducidad para asegurarse de que se eliminen los segmentos huérfanos.

Gestores de colas de nivel obsoleto

Cuando un gestor de colas con soporte para la segmentación genera un informe, pero se recibe en un gestor de colas que no tiene soporte para la segmentación, la estructura MQMDE (que identifica los campos *Offset* y *OriginalLength* representados por el informe) también se incluye en los datos de informe, además de cero, 100 bytes o todos los datos originales del mensaje.

No obstante, si un segmento de un mensaje pasa a través de un gestor de colas que no admite la segmentación, si se genera ahí un informe, la estructura MQMDE del mensaje original se trata puramente como datos. Por lo tanto, no se incluye en los datos de informe si se han solicitado cero bytes de los datos originales. Sin MQMDE, el mensaje de informe podría no resultar de utilidad.

Solicite al menos 100 bytes de datos en los informes si existe la posibilidad de que el mensaje pueda viajar a través de un gestor de colas de nivel obsoleto.

Formato de la información de control y los datos del mensaje

El gestor de colas solo está interesado en el formato de la información de control dentro de un mensaje, mientras que las aplicaciones que manejan el mensaje están interesadas en el formato de la información de control y de los datos.

Formato de la información de control del mensaje

La información de control en los campos de serie de caracteres del descriptor de mensaje debe estar en el juego de caracteres utilizado por el gestor de colas.

El atributo **CodedCharSetId** del objeto del gestor de colas define este juego de caracteres. La información de control debe estar en este juego de caracteres porque, cuando las aplicaciones pasan mensajes de un gestor de colas a otro, los agentes de canal de mensajes que transmiten los mensajes utilizan el valor de este atributo para determinar la conversión de datos que se va a realizar.

Formato de los datos del mensaje

Puede especificar cualquiera de los siguientes:

- El formato de los datos de aplicación
- El juego de caracteres de los datos de caracteres
- El formato de los datos numéricos

Para ello, utilice estos campos:

Format
Indica al destinatario de un mensaje el formato de los datos de aplicación en el mensaje.

Cuando el gestor de colas crea un mensaje, en algunas circunstancias utiliza el campo *Format* para identificar el formato de ese mensaje. Por ejemplo, cuando un gestor de colas no puede entregar un mensaje, coloca el mensaje en una cola de mensajes no entregados. Añade una cabecera (que contiene más información de control) al mensaje y cambia el campo *Format* para mostrarlo.

El gestor de colas tiene un número de *formatos incorporados* con nombres que empiezan MQ, por ejemplo, MQFMT_STRING. Si estos no satisfacen sus necesidades, puede definir sus propios formatos (*formatos definidos por el usuario*), pero no debe utilizar nombres que empiezan por MQ para ellos.

Cuando crea y utiliza sus propios formatos, debe escribir una salida de conversión de datos para dar soporte a un programa que obtiene el mensaje utilizando MQGMO_CONVERT.

CodedCharSetId
Define el juego de caracteres de los datos de caracteres en el mensaje. Si desea establecer este juego de caracteres en el de gestor de colas, puede establecer este campo en la constante MQCCSI_Q_MGR o MQCCSI_INHERIT.

Cuando obtenga un mensaje de una cola, compare el valor del campo *CodedCharSetId* con el valor que está esperando su aplicación. Si los dos valores difieren, es posible que tenga que convertir datos de caracteres en el mensaje o utilizar una salida de mensaje de conversión de datos si hay uno disponible.

Encoding
Describe el formato de los datos de los mensajes numéricos que contienen números enteros binarios, números enteros de decimales empaquetados y números de coma flotante. Normalmente, se codifica de acuerdo con la máquina específica en la que se ejecuta el gestor de colas.

Cuando pone un mensaje en una cola, normalmente especifica la constante MQENC_NATIVE en el campo *Encoding*. Esto significa que la codificación de los datos del mensaje es la misma que la de la máquina en la que se ejecuta la aplicación.

Cuando obtenga un mensaje de una cola, compare el valor del campo *Encoding* del descriptor de mensaje con el valor de la constante MQENC_NATIVE en la máquina. Si los dos valores difieren, es posible que tenga que convertir datos numéricos en el mensaje o utilizar una salida de mensaje de conversión de datos si hay uno disponible.

Conversión de datos de aplicación

Es posible que los datos de aplicación de deban convertir al conjunto de caracteres y la codificación que requiera otra aplicación, cuando se trata de plataformas diferentes.

Se puede convertir en el gestor de colas emisor o en el gestor de colas receptor. Si la biblioteca de formatos incluidos no se ajusta a sus necesidades, puede definir la suya propia. El tipo de conversión depende del formato del mensaje que se ha especificado en el campo de formato en el descriptor de mensaje, MQMD.

Nota: Los mensajes que tienen especificado MQFMT_NONE no se convierten.

Conversión en el gestor de colas emisor

Establezca el atributo de canal CONVERT en YES si necesita que el agente de canal de mensajes emisor (MCA) convierta los datos de aplicación.

La conversión se realiza en el gestor de colas emisor para determinados formatos incluidos para los formatos definidos por el usuario, si se proporciona una salida de usuario adecuada.

Formatos incluidos

Incluyen los siguientes:

- Mensajes que constan de caracteres en su totalidad (utilizando el nombre de formato MQFMT_STRING)
- Mensajes definidos por IBM MQ, por ejemplo, formatos de mandatos programables

  IBM MQ utilice los mensajes de formato de mandatos programables para mensajes y eventos de administración, en este caso, se utiliza el nombre de formato MQFMT_ADMIN. Puede utilizar el mismo formato, (utilizando el nombre de formato MQFMT_PCF), para sus propios mensajes y beneficiarse de la conversión incorporada.

Todos los formatos de gestor de colas incluidos tienen nombres que comienzan por MQFMT. En la sección Formato.

Formatos definidos por aplicación

Para los formatos definidos por usuario, la conversión de los datos de aplicación se debe realizar mediante un programa de salida de conversión de datos. Para obtener más información, consulte la sección “Escribir salidas de conversión de datos” en la página 1009. En un entorno de servidor y cliente, la salida se carga en el servidor y la conversión se lleva a cabo allí.

Conversión en el gestor de colas de recepción

Los datos del mensaje de aplicación los puede convertir el gestor de colas de recepción para los formatos incluidos y definidos por el usuario.

La conversión se realiza durante el proceso de una llamada MQGET, si especifica la opción MQGMO_CONVERT. Para obtener detalles, consulte Opciones

Juegos de caracteres codificados

Los productos IBM MQ dan soporte a los juegos de caracteres codificados que proporciona el sistema operativo subyacente.

Cuando crea un gestor de colas, se utiliza el ID de juego de caracteres codificados (CCSID) del gestor de colas, en función del entorno subyacente. Si se trata de una página de código mixto, IBM MQ utiliza la parte SBCS de la página de código mixto como el CCSID del gestor de colas.

En el caso de la conversión de datos general, si el sistema operativo subyacente da soporte a páginas de códigos DBCS, IBM MQ puede utilizarla.

Consulte la documentación de su sistema operativo para obtener detalles acerca de los conjuntos de caracteres codificados a los que da soporte.

Debe tener en cuenta la conversión de datos de aplicación, los nombres de formatos y las salidas de usuario cuando escriba aplicaciones que abarcan varias plataformas. Consulte la sección “Escribir salidas

de conversión de datos” en la página 1009 para obtener información acerca de cómo invocar las salidas de conversión de datos.

Prioridades de mensajes

Puede establecer la prioridad de un mensaje en un valor numérico, o dejar que el mensaje tome la prioridad predeterminada de la cola.

Establece la prioridad de un mensaje (en el campo *Priority* de la estructura MQMD) cuando coloca el mensaje en una cola. Puede establecer un valor numérico para la prioridad, o puede dejar que el mensaje tome la prioridad predeterminada de la cola.

El atributo **MsgDeliverySequence** de la cola determina si los mensajes contenidos en la cola se almacenan según el orden FIFO (primero en entrar, primero en salir) o en el orden FIFO dentro de cada nivel de prioridad. Si este atributo se establece en MQMDS_PRIORITY, los mensajes se ponen en cola con la prioridad especificada en el campo *Priority* de sus descriptores de mensaje; pero si se establece en MQMDS_FIFO, los mensajes se ponen en cola con la prioridad predeterminada de la cola. Los mensajes de igual prioridad se almacenan en la cola por orden de llegada.

El atributo **DefPriority** de una cola establece el valor de prioridad predeterminado para los mensajes que se colocan en esa cola. Este valor se establece cuando se crea la cola, pero se puede cambiar posteriormente. Las colas de alias y las definiciones locales de las colas remotas pueden tener prioridades predeterminadas diferentes de las colas base resultantes de la resolución de aquéllas. Si hay más de una definición de cola en la vía de acceso de resolución (consulte “Resolución de nombres” en la página 764), la prioridad predeterminada se toma del valor (en el momento de la operación de colocación) del atributo **DefPriority** de la cola especificada en el mandato open.

El valor del atributo **MaxPriority** del gestor de colas es la prioridad máxima que puede asignar a un mensaje procesado por ese gestor de colas. No puede modificar el valor de este atributo. En IBM MQ, el atributo tiene el valor 9; puede crear mensajes que tengan prioridades entre 0 (la más baja) y 9 (la más alta).

Propiedades del mensaje

Utilice las propiedades de mensaje para permitir que una aplicación seleccione mensajes para procesar o para recuperar información sobre un mensaje sin acceder a las cabeceras MQMD o MQRFH2. Las propiedades de mensaje también facilitan la comunicación entre las aplicaciones IBM MQ y JMS.

Una propiedad de mensaje es datos asociados a un mensaje, que consta de un nombre textual y un valor de un tipo determinado. Las propiedades de mensaje son utilizadas por los selectores de mensajes para filtrar publicaciones de acuerdo con temas o para obtener de forma selectiva mensajes de las colas. Las propiedades de mensaje se pueden utilizar para incluir datos empresariales o información de estado sin tener que almacenarlos en los datos de aplicación. Las aplicaciones no necesitan acceder a datos contenidos en las cabeceras MQMD (MQ Message Descriptor) o MQRFH2, pues los campos de estas estructuras de datos se pueden acceder como propiedades de mensaje utilizando llamadas de función de la interfaz de cola de mensajes (MQI).

El uso de las propiedades de mensaje en IBM MQ se asemeja al uso de las propiedades en JMS. Esto significa que puede establecer propiedades en una aplicación JMS y recuperarlas en una aplicación de procedimiento de IBM MQ, o a la inversa. Para hacer que una propiedad sea accesible para una aplicación de JMS, asigne el prefijo "usr" a la propiedad; entonces estará disponible (sin el prefijo) como propiedad de usuario de mensaje de JMS. Por ejemplo, la propiedad de IBM MQ *usr.myproperty* (una serie de caracteres) es accesible para una aplicación JMS utilizando la llamada JMS `message.getStringProperty('myproperty')`. Observe que las aplicaciones de JMS no pueden acceder a propiedades que tengan el prefijo "usr" si contienen dos o más caracteres U+002E ("."). Una propiedad sin prefijo y ningún carácter de U + 002E (".") se trata como si tuviera el prefijo "usr". Por el contrario, se puede acceder a una propiedad de usuario establecida en una aplicación JMS en una aplicación IBM MQ añadiendo el "usr." al nombre de propiedad consultado en una llamada MQINQMP.

Propiedades y longitud de un mensaje

Utilice el atributo del gestor de colas *MaxPropertiesLength* para controlar el tamaño de las propiedades que pueden fluir con cualquier mensaje en un gestor de colasIBM MQ.

En general, cuando se usa MQSETMP para definir propiedades, el tamaño de una propiedad es la longitud de su nombre en bytes más la longitud de su valor en bytes tal y como se pasan a la llamada MQSETMP. Es posible que el juego de caracteres del nombre y del valor de la propiedad cambie durante la transmisión del mensaje a su destino, porque estos pueden convertirse a Unicode; en este caso, el tamaño de la propiedad puede cambiar.

En una llamada MQPUT o MQPUT1, las propiedades del mensaje no cuentan en lo que respecta a la longitud del mensaje de la cola y el gestor de colas, pero sí cuentan en lo que respecta a las propiedades tal como las percibe el gestor de colas (tanto si se han establecido con llamadas de propiedad de mensaje MQI o no).

Si el tamaño de las propiedades supera la longitud máxima de las propiedades, el mensaje se rechaza con MQRC_PROPERTIES_TOO_BIG. Puesto que el tamaño de las propiedades depende de su representación, hay que establecer la longitud máxima de propiedades de forma gruesa.

Es posible que una aplicación pueda colocar correctamente un mensaje con un búfer que supere el valor de *MaxMsgLength*, si el búfer incluye propiedades. Esto se debe a que, incluso cuando se representan como elementos de MQRFH2, las propiedades de mensaje no cuentan para la longitud del mensaje. Los campos de la cabecera MQRFH2 se suman a la longitud de las propiedades solo si hay una o más carpetas contenidas y cada una de dichas carpetas en la cabecera contienen propiedades. Si una o más carpetas están contenidas en la cabecera MQRFH2 y hay alguna carpeta que no contenga propiedades, los campos de cabecera MQRFH2 contarán para la longitud del mensaje en su lugar.

En una llamada MQGET, las propiedades del mensaje no cuentan para la longitud del mensaje en lo que respecta a la cola y al gestor de colas. Sin embargo, puesto que las propiedades se cuentan de forma separada, puede que el búfer devuelto por una llamada MQGET supere el valor del atributo *MaxMsgLength*.

No haga que las aplicaciones consulten el valor de *MaxMsgLength* y luego asignen un búfer de ese tamaño antes de invocar MQGET; en su lugar, asigne un búfer que le parezca lo suficientemente grande. Si la MQGET falla, asigne búfer tomando como referencia el tamaño del parámetro *DataLength*.

El parámetro *DataLength* de la llamada MQGET devuelve la longitud en bytes de los datos de la aplicación y de las propiedades devueltas en el búfer proporcionado, si no se especifica un descriptor de mensajes en la estructura MQGMO.

El parámetro *Buffer* de la llamada MQPUT contiene los datos de mensaje de aplicación que se van a enviar y las propiedades representadas en los datos del mensaje.

Hay un límite de longitud de 100 MB para las propiedades del mensaje, excluyendo el descriptor de mensaje o la extensión de cada mensaje.

El tamaño de una propiedad en su representación interna es la longitud del nombre más el tamaño de su valor más algunos datos de control de dicha propiedad. También hay algunos datos de control para el conjunto de propiedades después de añadirse una propiedad al mensaje.

Nombres de propiedades

Un nombre de propiedad es una serie de caracteres. Se aplican determinadas restricciones a su longitud y al juego de caracteres que puede utilizarse.

Un nombre de propiedad es un serie de caracteres sensible a mayúsculas y minúsculas, limitada a +4095 caracteres, a menos que esté restringido por el contexto. Este límite se encuentra en la constante MQ_MAX_PROPERTY_NAME_LENGTH.

Si excede esta longitud máxima cuando se utiliza una llamada MQI de propiedad de mensaje, la llamada falla con el código de razón MQRC_PROPERTY_NAME_LENGTH_ERR.

Como no hay ninguna longitud máxima de nombre de propiedad en JMS, es posible que una aplicación JMS establezca un nombre de propiedad JMS válido que no sea un nombre de propiedad de IBM MQ válido cuando se almacena en una estructura MQRFH2.

En este caso, cuando se analiza, solo se utilizan los primeros 4095 caracteres del nombre de propiedad; los caracteres siguientes se truncan. Esto puede hacer que una aplicación que utiliza selectores no coincida con una serie de selección, o que coincida con una serie cuando no se esperaba, ya que varias propiedades podrían truncarse con el mismo nombre. Cuando se trunca un nombre de propiedad, WebSphere MQ emite un mensaje de registro de errores.

Todos los nombres de propiedad deben seguir las reglas definidas por la especificación de lenguaje Java para los identificadores Java, con la excepción de que se permite el carácter Unicode U+002E (.) como parte del nombre, pero no al principio. Las reglas de los identificadores Java equivalen a las contenidas en la especificación JMS para los nombres de propiedad.

Los caracteres de espacio en blanco y los operadores de comparación están prohibidos. Se permiten nulos intercalados en un nombre de propiedad, pero no se recomiendan. Si utiliza nulos intercalados, no se puede utilizar la constante MQVS_NULL_TERMINATED cuando se utiliza con la estructura MQCHARV para especificar series de longitud variable.

Mantenga los nombres de propiedad simples porque las aplicaciones pueden seleccionar mensajes en función de los nombres de propiedad y la conversión entre el juego de caracteres del nombre y del selector puede hacer que la selección falle inesperadamente.

Los nombres de propiedad de IBM MQ utilizan el carácter U+002E (.) para la agrupación lógica de propiedades. Esto divide el espacio de nombres para las propiedades. Las propiedades con los prefijos siguientes, en cualquier combinación de minúsculas o mayúsculas, están reservadas para su uso en el producto:

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

Una buena forma de evitar conflictos de nombre es garantizar que todas las aplicaciones añadan su nombre de dominio de Internet como prefijo en las propiedades de mensaje. Por ejemplo, si está desarrollando una aplicación utilizando el nombre de dominio `ourcompany.com`, puede nombrar todas las propiedades con el prefijo `com.ourcompany`. Este convenio de denominación también permite una selección más fácil de las propiedades, por ejemplo, una aplicación puede consultar todas las propiedades de mensaje que empiezan por `com.ourcompany.%`.

Consulte Restricciones de nombre de propiedad para obtener más información sobre el uso de nombres de propiedad.

Restricciones para nombres de propiedad
Cuando asigna un nombre a una propiedad, debe respetar determinadas reglas.

Se aplican las restricciones siguientes a los nombres de propiedad:

1. Un nombre de propiedad no puede comenzar con las cadenas de caracteres siguientes:
   - "JMS" - su uso está reservado para IBM MQ classes for JMS.
   - "usr.JMS" - no es válido.

   Las únicas excepciones son las propiedades siguientes que proporcionan sinónimos para propiedades de JMS:

Propiedad	**Sinónimo de**
JMSCorrelationID	Root .MQMD.CorrelId o jms.Cid

Propiedad	Sinónimo de
JMSDeliveryMode	Root .MQMD.Persistence o jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root .MQMD.Expiry o jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root .MQMD.Priority o jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo (cadena de caracteres codificada como un URI)	Root .MQMD.ReplyToQ o Root .MQMD.ReplyToQMgr o jms.Rto
JMSTimestamp	Root .MQMD.PutDate o Root .MQMD.PutTime o jms.Tms
JMSType	mcd.Type o mcd.Set o mcd.Fmt
JMSXAppID	Root .MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root .MQMD.GroupId o jms.Gid
JMSXGroupSeq	Root .MQMD.MsgSeqNumber o jms.Seq
JMSXUserID	Root .MQMD.UserIdentifier

Estos sinónimos permiten que una aplicación de MQI acceda a propiedades de JMS de forma similar a la aplicación cliente de IBM MQ classes for JMS. De estas propiedades, sólo JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID y JMSXGroupSeq se pueden establecer utilizando la interfaz MQI.

Observe que las propiedades JMS_IBM_* disponibles desde dentro de IBM MQ classes for JMS no están disponibles cuando se utiliza MQI. Los campos a los que hacen referencia las propiedades JMS_IBM_* pueden ser accedidos de otras maneras por las aplicaciones MQI.

2. El nombre de una propiedad no puede ser "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" ni "ESCAPE", para cualquiera que sea la combinación utilizada de mayúsculas y minúsculas. Esos nombres son palabras clave de SQL que se utilizan en cadenas de selección.
3. Un nombre de propiedad que empieza por " mq " en cualquier combinación de mayúsculas y minúsculas y no empieza por "mq_usr" solo puede contener un "." (U+002E). Múltiple "." no están permitidos en las propiedades con esos prefijos.
4. Dos "." los caracteres deben contener otros caracteres en medio; no puede tener un punto vacío en la jerarquía. De forma similar, un nombre de propiedad no puede terminar en un "." .
5. Si una aplicación define la propiedad "a.b" y luego la propiedad "a.b.c", no está claro si en la jerarquía "b" contiene un valor u otra agrupación lógica. Esta es una jerarquía de "contenido mixto" y no está permitida. No está permitido definir una propiedad que produzca contenido contenido mixto.

El mecanismo de validación aplica estas restricciones de la forma siguiente:

- Los nombres de propiedad se validan al establecer una propiedad utilizando la llamada MQSETMP- Establezca la propiedad de mensaje, si se ha solicitado validación al crear el descriptor de contexto de mensaje. Si la validación de una propiedad falla debido a un error en la especificación del nombre de propiedad, el código de terminación es MQCC_FAILED y el código de razón devuelto es:
  - MQRC_PROPERTY_NAME_ERROR para las razones 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED para la razón 5.
- Los nombres de propiedades especificadas directamente como elementos MQRFH2 no es seguro que se validen mediante la llamada MQPUT.

Campos del descriptor de mensaje como propiedades
La mayoría de los campos del descriptor de mensaje pueden tratarse como propiedades. El nombre de propiedad se construye añadiendo un prefijo al nombre del campo del descriptor de mensaje.

Si una aplicación MQI desea identificar una propiedad de mensaje contenida en un campo de descriptor de mensaje, por ejemplo, en una serie de selector o utilizando las API de propiedades de mensaje, utilice la siguiente sintaxis:

Nombre de propiedad	Campo del descriptor de mensaje
Root.MQMD.*Field*	*Campo*

Especifique *Field* con las mismas mayúsculas y minúsculas que para los campos de estructura MQMD en la declaración de lenguaje C. Por ejemplo, el nombre de propiedad `Root.MQMD.AccountingToken` accede al campo `AccountingToken` del descriptor de mensaje.

Los campos `StrucId` y `Version` del descriptor de mensaje no son accesibles utilizando la sintaxis mostrada.

Los campos de descriptor de mensaje nunca se representan en una cabecera MQRFH2 como para otras propiedades.

Si los datos del mensaje empiezan por un MQMDE respetado por el gestor de colas, se puede acceder a los campos MQMDE utilizando la notación `Root.MQMD.`*`Field`* que se describe. En este caso, los campos MQMDE se tratan como una parte lógica de MQMD desde la perspectiva de las propiedades. Consulte Visión general de MQMDE.

Tipos de datos y valores de una propiedad
Una propiedad puede ser un booleano, una cadena de bytes, una cadena de caracteres o un número en coma flotante o entero. La propiedad puede almacenar cualquier valor válido en el rango del tipo de datos a menos que el contexto imponga otras restricciones.

El tipo de datos de un valor de propiedad tiene que ser uno de los valores siguientes:

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

Una propiedad puede existir sin tener un valor definido: se trata de una propiedad nula. Una propiedad nula se diferencia de una propiedad de byte (MQBYTE []) o de una propiedad de cadena de caracteres (MQCHAR []) en el que tiene un valor definido pero vacío, es decir, uno con un valor de longitud cero.

La cadena de bytes no es un tipo de datos de propiedad válido en JMS ni XMS. Se recomienda no usar propiedades de cadena de bytes en la carpeta *usr*.

Selección de mensajes de una cola

Se pueden seleccionar mensajes de una cola usando los campos MsgId y CorrelId en una llamada MQGET, o bien con una SelectionString en una llamada MQOPEN o MQSUB.

Selectores
Un selector de mensajes es una cadena de longitud variable que una aplicación utiliza para registrar su interés en aolo aquellos mensajes que tengan propiedades que respondan a consulta SQL (lenguaje de consulta estructurado) que la cadena de la selección representa.

Selección utilizando las llamadas de función MQSUB y MQOPEN

Utilice *SelectionString*, que es una estructura de tipo MQCHARV, para efectuar selecciones mediante las llamadas MQSUB y MQOPEN.

La estructura *SelectionString* se utiliza para pasar una cadena de selección de longitud variable al gestor de colas.

El CCSID asociado a la cadena del selector se establece a través del campo VSCCSID de la estructura MQCHARV. El valor utilizado tiene que ser un CCSID que esté soportado en cadenas de selector. Consulte Conversión de página de códigos para obtener la lista de páginas de códigos soportadas.

La especificación de un CCSID para el cual no hay ninguna conversión Unicode soportada de IBM MQ genera un error de MQRC_SOURCE_CCSID_ERROR. Este error se devuelve en el momento en que el selector se presenta al gestor de colas, es decir, en las llamadas MQSUB, MQOPEN o MQPUT1.

El valor predeterminado del campo *VSCCSID* es MQCCSI_APPL, que indica que el CCSID de la cadena de selección es igual que el CCSID del gestor de colas, o que el CCSID del cliente, si se ha conectado a través de un cliente. Sin embargo, la constante MQCCSI_APPL puede ser sustituida por una aplicación que la redefina antes de compilar.

Si el selector MQCHARV representa una cadena NULL, no se efectúa ninguna selección para dicho consumidor de mensajes y los mensajes se entregan como si no se hubiera utilizado un selector.

La longitud máxima de una cadena de selección solo está limitada por lo que se pueda describir en el campo *VSLength* de MQCHARV.

Se devuelve SelectionString en la salida de una llamada MQSUB utilizando la opción de suscripción MQSO_RESUME si ha proporcionado un búfer y hay una longitud de búfer positiva en VSBufSize. Si no se proporciona un búfer, en el campo VSLength de MQCHARV solo se devuelve la longitud de la cadena de selección. Si el búfer proporcionado es inferior al espacio necesario para devolver el campo, solo se devuelven VSBufSize bytes en el búfer.

Una aplicación no puede modificar una cadena de selección sin cerrar antes el descriptor de cola (en MQOPEN) o de suscripción (en MQSUB). A continuación, se puede especificar una cadena de selección nueva en llamadas MQSUB o MQOPEN posteriores.

MQOPEN
Utilice MQCLOSE para cerrar el descriptor de contexto abierto y luego especifique una nueva cadena de selección en una llamada MQOPEN posterior.

MQSUB
Utilice MQCLOSE para cerrar el descriptor de suscripción devuelto (hsub) y luego especifique una nueva cadena de selección en una llamada MQSUB posterior.

Figura 3 en la página 33 muestra el proceso de selección utilizando la llamada MQSUB.

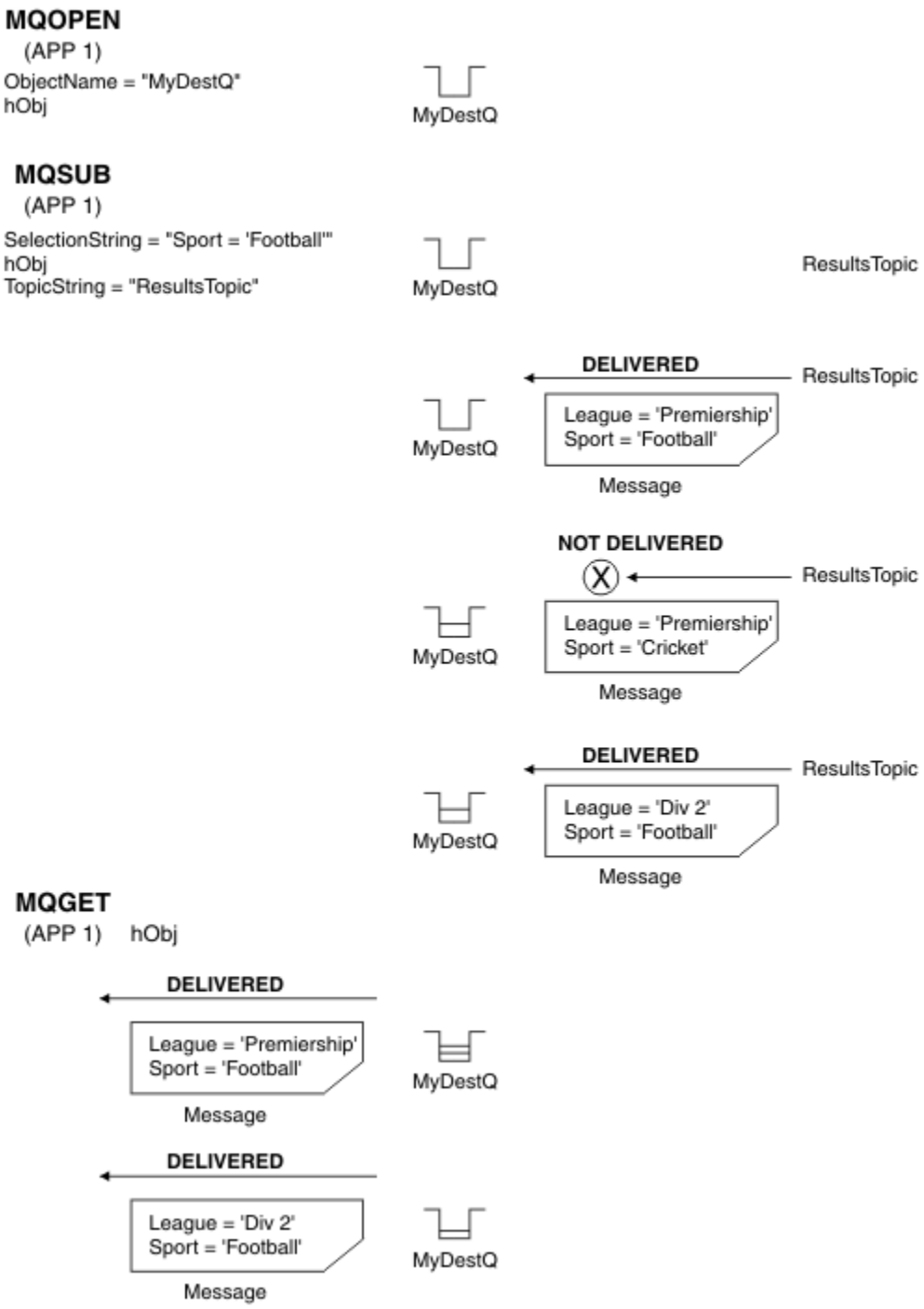


Figura 3. Selección con una llamada MQSUB

Se puede pasar un selector en la llamada a MQSUB utilizando el campo *SelectionString* en la estructura MQSD. El efecto de pasar un selector en MQSUB es que en la cola de destino solo estarán disponibles los mensajes publicados en el tema al que se esté suscrito y que coincidan con la cadena de suscripción proporcionada.

Figura 4 en la página 34 muestra el proceso de selección utilizando la llamada MQOPEN.

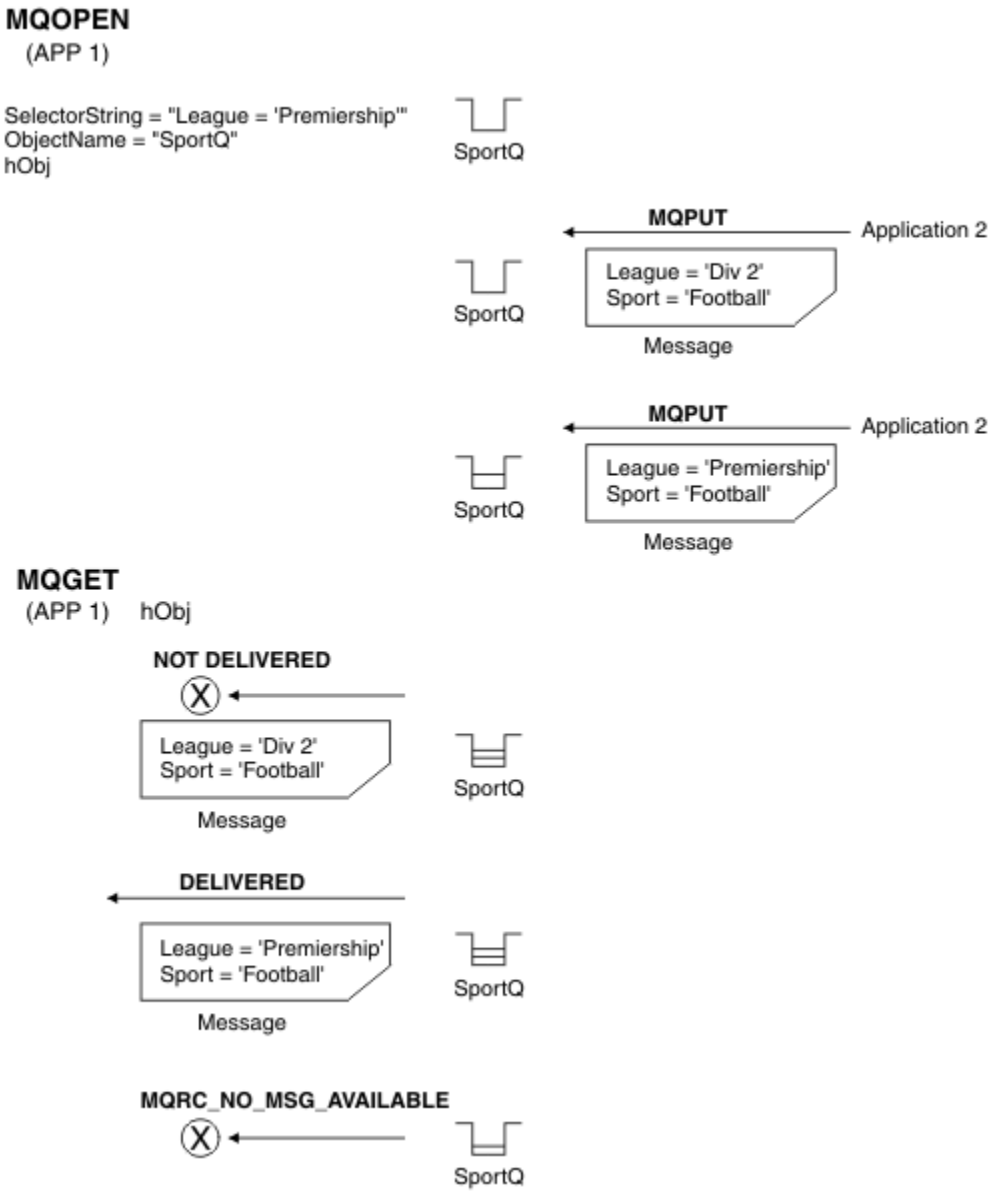


Figura 4. Selección con la llamada MQOPEN

Se puede pasar un selector en la llamada a MQOPEN utilizando el campo *SelectionString* en la estructura MQOD. El efecto de pasar en un selector en la llamada MQOPEN es que solo se entregan al consumidor de mensajes aquellos mensajes de la cola abierta que coinciden con un selector.

El uso principal del selector en una llamada MQOPEN se da en el caso punto a punto donde una aplicación puede optar por recibir solo aquellos mensajes de una cola que coincidan con un selector. El ejemplo anterior se muestra un escenario simple en el que se colocan dos mensajes en una cola abierta por MQOPEN, pero la aplicación que hace la obtención solo recibe uno, el que coincide con el selector.

Tenga en cuenta que las llamadas posteriores de MQGET dan como resultado MQRC_NO_MSG_AVAILABLE, ya que no existen mensajes adicionales en la cola que coincidan con el selector facilitado.

Conceptos relacionados

“Restricciones y reglas de series de selección” en la página 41
Familiarícese con estas reglas acerca de cómo se interpretan las series de selección y las restricciones para evitar problemas potenciales cuando se utilizan selectores.

Comportamiento de la selección

Descripción general del comportamiento de la selección en IBM MQ

Los campos de una estructura MQMDE se considerarán las propiedades de mensaje de las correspondientes propiedades del descriptor de mensaje si el MQMD:

- Tiene el formato MQFMT_MD_EXTENSION.
- Está seguido inmediatamente de una estructura MQMDE válida.
- Es de versión uno o contiene la versión predeterminada solo en dos campos.

Es posible resolver una cadena de selección a TRUE o a FALSE antes de que se lleve a cabo cualquier comparación con las propiedades del mensaje. Por ejemplo, podría ser el caso si la serie de selección se establece en `"TRUE <> FALSE"`. Esta evaluación temprana solo se garantiza cuando no hay ninguna referencia a propiedad de mensaje en la cadena de selección.

Si una cadena de selección se resuelve a TRUE antes de considerarse cualquier propiedad de mensaje, se entregarán todos los mensajes publicados en el tema al que se haya suscrito el consumidor. Si una cadena de selección se resuelve a FALSE antes de considerarse cualquier propiedad de mensaje, se devolverán el código de razón de MQRC_SELECTOR_ALWAYS_FALSE y el código de terminación MQCC_FAILED en la llamada de función que ha presentado el selector.

Incluso si un mensaje no contiene ninguna propiedad de mensaje (aparte de las propiedades de cabecera), todavía puede ser elegible para la selección. Si una cadena de selección referencia una propiedad de mensaje inexistente, se asume que dicha propiedad tiene el valor NULL o 'Unknown'.

Por ejemplo, un mensaje todavía puede cumplir una cadena de selección como, por ejemplo, `'Color IS NULL'`, donde `'Color'` no existe como propiedad de mensaje en el mensaje.

La selección solo se puede realizar en las propiedades asociadas a un mensaje, no en el propio mensaje, a menos que esté disponible un proveedor de selecciones de mensajes ampliado. La selección solo se puede realizar en la carga útil del mensaje si hay disponible un proveedor de selecciones de mensajes ampliado.

Cada propiedad de mensaje tiene asociado un tipo. Cuando se realiza una selección, hay que asegurarse de que los valores utilizados en las expresiones para comprobar las propiedades de mensaje sean del tipo correcto. Si se produce una discordancia de tipos, la expresión en cuestión resolverá a FALSE.

Es responsabilidad de usted asegurarse de que la cadena de selección y las propiedades de mensaje usen tipos compatibles.

Los criterios de selección se siguen aplicando en nombre de los suscriptores duraderos inactivos, de modo que solo se conservan los mensajes que coinciden con la cadena de selección suministrada originalmente.

Las cadenas de selección no son modificables cuando se reanuda una suscripción duradera con alteración (MQSO_ALTER). Si se presenta una cadena de selección distinta cuando un suscriptor duradero reanuda la actividad, se devuelve MQRC_SELECTOR_NOT_ALTERABLE a la aplicación.

Las aplicaciones reciben el código de retorno MQRC_NO_MSG_AVAILABLE si no hay ningún mensaje en una cola que cumpla los criterios de selección.

Si una aplicación ha especificado una cadena de selección que contiene valores de propiedad, solo los mensajes que contengan propiedades coincidentes serán elegibles para la selección. Por ejemplo, un suscriptor especifica una serie de selección de "a = 3" y se publica un mensaje que no contiene propiedades, o propiedades donde 'a' no existe o no es igual a 3. El suscriptor no recibe ese mensaje en su cola de destino.

Rendimiento de la mensajería

La selección de mensajes de una cola requiere que IBM MQ inspeccione secuencialmente cada mensaje en la cola. Los mensajes se inspeccionan hasta que se encuentra un mensaje que coincide con los criterios de selección o no haya más mensajes por examinar. Por lo tanto, el rendimiento de la mensajería se ve penalizado si se utiliza la selección de mensajes en colas profundas.

Para optimizar la selección de mensajes en colas profundas cuando la selección se basa en JMSCorrelationID o JMSMessageID, utilice una serie de selección con el formato:

- JMSCorrelationID = 'ID:*ID_correlación*'
- JMSMessageID= 'ID:*ID_mensaje*'

donde:

- *correlation_id* es un valor String que contiene un identificador de correlación IBM MQ estándar.
- *message_id* es un valor String que contiene un identificador de mensaje IBM MQ estándar.

Nota: El selector sólo debe hacer referencia a una de las propiedades. El uso de un selector que tiene uno de estos formatos ofrece una mejora significativa en el rendimiento al seleccionar en JMSCorrelationID y ofrece una mejora marginal en el rendimiento para JMSMessageID. Para obtener más información, consulte "Selectores de mensajes en JMS" en la página 150.

Uso de selectores complejos

Los selectores pueden contener muchos componentes, por ejemplo:

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

El uso de estos selectores complejos puede penalizar seriamente el rendimiento y plantear excesivos requisitos de recursos. Por tanto, IBM MQ protege el sistema no procesando selectores excesivamente complejos que podrían provocar una escasez de recursos del sistema. La protección se puede producir en las series de selección que contienen más de 100 pruebas o cuando IBM MQ detecta que se está alcanzando el límite sobre el tamaño de la pila del sistema operativo. Hay que probar a fondo el uso de cadenas de selección con muchos componentes, en las correspondients plataformas, para asegurarse de que no se alcanzan los límites de protección.

El rendimiento y la complejidad de los selectores se pueden mejorar simplificándolos utilizando paréntesis adicionales para combinar componentes. Por ejemplo:

```
( a y b o c y d) o ( e y f o g y h) o ( i y j) ...
```

Conceptos relacionados

"Restricciones y reglas de series de selección" en la página 41
Familiarícese con estas reglas acerca de cómo se interpretan las series de selección y las restricciones para evitar problemas potenciales cuando se utilizan selectores.

Sintaxis del selector de mensajes

Un selector de mensajes de IBM MQ es una serie de caracteres con sintaxis que se basa en un subconjunto de la sintaxis de expresiones condicionales SQL92.

El orden en el que se evalúa un selector de mensajes es de izquierda a derecha dentro de un nivel de prioridad. Se pueden utilizar paréntesis para cambiar este orden. Los literales de selector y nombres de operador predefinidos se presentan aquí escritos en mayúsculas, pero no hay distinción entre mayúsculas y minúsculas.

Si el selector se proporciona a través de la API, IBM MQ verifica la corrección sintáctica de un selector de mensajes en el momento en que se presenta. Si la sintaxis de la serie de selección es incorrecta o un nombre de propiedad no es válido y no está disponible un proveedor de selección de mensajes ampliados, se devuelve `MQRC_SELECTION_NOT_AVAILABLE` a la aplicación. Si la sintaxis de la serie de selección es incorrecta o un nombre de propiedad no es válido cuando se reanuda una suscripción, se devuelve un `MQRC_SELECTOR_SYNTAX_ERROR` a la aplicación. Si la validación de nombres de propiedad se ha inhabilitado (mediante el establecimiento de `MQCMHO_NONE` en lugar de `MQCMHO_VALIDATE`) y una aplicación coloca posteriormente un mensaje con un nombre de propiedad no válido, este mensaje no se selecciona nunca.

No se devuelve ningún error en el momento en que se presenta el selector si IBM MQ determina que un selector de suscripción definido administrativamente está utilizando sintaxis de mensaje ampliado, tal como indica el parámetro **DISPLAY SUB SELTYPE** que tiene el valor EXTENDED. En este caso, la

comprobación de la sintaxis de la serie de selección se aplaza hasta la hora de publicación (consulte MQRC_SELECTION_NOT_AVAILABLE).

Un selector puede contener:

- Literales:
  - Los literales de tipo serie están encerrados entre comillas simples. Dos comillas simples consecutivas representan una sola comilla. Ejemplos: 'literal' y 'literal''s'. Como en el caso de los literales de tipo serie de Java, se utiliza la codificación de caracteres Unicode. No puede utilizar comillas dobles para encerrar un literal de tipo serie. Se puede utilizar cualquier secuencia de bytes entre las comillas simples.
  - Una serie de bytes consta de uno o más pares de caracteres hexadecimales encerrados entre comillas dobles y precedidos por 0x. Ejemplos: "0x2F1C", "0XD43A". La longitud de una serie de bytes debe ser como mínimo de un byte. Si una serie de bytes de selector se asocia con una propiedad de mensaje de tipo MQTYPE_BYTE_STRING, no se ejecuta ninguna acción especial en el cero inicial o final. Los bytes se tratan como otro carácter. Tampoco se hace distinción entre los formatos Little Endian y Big Endian. La longitud de la serie del selector y de la serie de bytes de la propiedad debe ser igual, y la secuencia de bytes debe ser la misma.

    Ejemplos de selecciones de series de bytes coincidentes (se supone que *myBytes* = 0AFC23):

    - "myBytes = "0x0AFC23"" = TRUE

    Las selecciones de series siguientes no coinciden:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (porque el número de bytes no es múltiplo de dos)
    - "myBytes = "0x0AFC2300"" = FALSE (porque el cero final es significativo en la comparación)
    - "myBytes = "0x000AFC23"" = FALSE (porque el cero inicial es significativo en la comparación)
    - "myBytes = "0x23FC0A"" = FALSE (porque los formatos Little Endian y Big Endian no se tienen en cuenta)
  - Los números hexadecimales empiezan con un cero, seguido de un xen mayúsculas o minúsculas. El resto del literal contiene uno o más caracteres hexadecimales válidos. Ejemplos: 0xA, 0xAF, 0X2020.
  - Un cero inicial seguido de uno o más dígitos dentro del rango 0-7 se interpreta siempre como el inicio de un número octal. No puede representar un número decimal con prefijo cero; por ejemplo, 09 devuelve un error de sintaxis porque 9 no es un dígito octal válido. Ejemplos de números octales: 0177, 0713.
  - Un literal numérico exacto es un valor numérico sin una coma decimal, tal como 57, -957 y +62. Un literal numérico exacto puede tener una L final en mayúscula o minúscula; esto no afecta a la forma en que se almacena o interpreta el número. IBM MQ da soporte a números exactos en el rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
  - Un literal numérico aproximado es un valor numérico en notación científica, como 7E3 o -57.9E2, o un valor numérico con un decimal, como 7., -95.7o +6.2. IBM MQ admite los números comprendidos de -1.797693134862315E+308 a 1.797693134862315E+308.

    El significante debe seguir un carácter de signo opcional (+ o -). El significante debe ser un entero o una fracción. Una parte fraccional del significante no es necesario que tenga un dígito inicial.

    Una E en mayúscula o minúscula indica el inicio de un exponente opcional. El exponente tiene una raíz decimal y la parte numérica del exponente puede está precedida opcionalmente por un carácter de signo.

    Los literales numéricos aproximados pueden terminar en un carácter F o D (sin distinción entre mayúsculas y minúsculas). Esta sintaxis existe para dar soporte al método de lenguaje cruzado de etiquetado de números de precisión simple o doble. Estos caracteres son opcionales y no afectan a la forma en que se almacena o procesa un literal numérico aproximado. Estos números siempre se almacenan y procesan utilizando la precisión doble.
  - Los literales booleanos TRUE y FALSE.

Nota: Las especificaciones IEEE-754 no finitas, tales como `NaN`, `+Infinity`, `-Infinity`, no están soportadas en las series de selección. Por lo tanto, no es posible utilizar estos valores como operandos en una expresión. El cero negativo es tratado igual que el cero positivo para las operaciones matemáticas.

- Identificadores:

  un identificador es una secuencia de caracteres de longitud variable que debe empezar con un carácter de inicio de identificador válido, seguido de cero o más caracteres de parte de identificador válidos. Las reglas para los nombres de identificador son las mismas que para los nombres de propiedad de mensaje, consulte "Nombres de propiedades" en la página 28 y "Restricciones para nombres de propiedad" en la página 29 para obtener más información.

  Nota: La selección solo se puede realizar en la carga útil del mensaje si hay disponible un proveedor de selecciones de mensajes ampliado.

  Los identificadores son referencias de campo de cabecera o referencias de propiedad. El tipo de un valor de propiedad en un selector de mensajes debe corresponder al tipo utilizado para establecer la propiedad, aunque la promoción numérica se lleva a cabo siempre que sea posible. Si se produce una falta de coincidencia de tipo, el resultado de la expresión es `FALSE`. Si se hace referencia a una propiedad que no existe en un mensaje, su valor es `NULL`.

  Las conversiones de tipo que se aplican a los métodos get para las propiedades no se aplican cuando se utiliza una propiedad en una expresión de selector de mensajes. Por ejemplo, si establece una propiedad como un valor de tipo serie y luego utiliza un selector para consultarlo como un valor numérico, la expresión devuelve `FALSE`.

  Los nombres de campo y de propiedad de JMS que se correlacionan con nombres de propiedad o nombres de campo de `MQMD` también son identificadores válidos en una serie de selección. IBM MQ correlaciona los nombres de campo y de propiedad reconocidos de JMS con los valores propiedad de mensaje. Para obtener más información, consulte "Selectores de mensajes en JMS" en la página 150. A modo de ejemplo, la serie de selección `"JMSPriority >="` selecciona de acuerdo con la propiedad `Pri` encontrada en la carpeta `jms` del mensaje actual.

- Desbordamiento/subdesbordamiento:

  Para los valores numéricos decimales y aproximados, las condiciones siguientes están sin definir:

  – Especificar un número que está fuera del rango definido
  – Especificar una expresión aritmética que produciría un desbordamiento o subdesbordamiento

  No se realizan comprobaciones para estas condiciones.

- Espacio en blanco:

  Definido como un espacio, alimentación de papel, línea nueva, retorno de carro, tabulador horizontal o tabulador vertical. Los caracteres Unicode siguientes se reconocen como espacio en blanco:

  – `\u0009 to \u000D`
  – `\u0020`
  – `\u001C`
  – `\u001D`
  – `\u001E`
  – `\u001F`
  – `\u1680`
  – `\u180E`
  – `\u2000` a `\u200A`
  – `\u2028`
  – `\u2029`
  – `\u202F`

  - \u205F
  - \u3000
- Expresiones:
  - Un selector es una expresión condicional. Un selector cuya evaluación da un resultado verdadero produce una coincidencia; un selector cuya evaluación da un resultado falso o desconocido no produce una coincidencia.
  - Las expresiones aritméticas se componen de sí mismas, operaciones aritméticas, identificadores (el valor de identificador se trata como un literal numérico) y literales numéricos.
  - Las expresiones condicionales se componen de sí mismas, operaciones de comparación y operaciones lógicas.
- Están permitidos los corchetes estándar () para establecer el orden en el que se evalúan las expresiones.
- Operadores lógicos en orden de prioridad: NOT, AND, OR.
- Operadores de comparación: =, >, >=, <, <=, <> (no igual).
  - Dos series de bytes son iguales sólo si las series tienen la misma longitud y la secuencia de bytes es igual.
  - Sólo se pueden comparar valores del mismo tipo, Una excepción es que es válido comparar valores numéricos exactos y valores numéricos aproximados (la conversión de tipo necesaria se define mediante las reglas de promoción numérica de Java). Si se intentan comparar tipos diferentes, el selector siempre es falso (false).
  - La comparación de series y la comparación booleana están restringidas a = y <>. Dos series sólo son iguales si contienen la misma secuencia de caracteres.
- Operadores aritméticos por orden de prioridad:
  - Operadores unarios +, -.
  - * multiplicación y / división.
  - + adición y - sustracción
  - No están permitidas las operaciones aritméticas sobre un valor NULL. Si se intentan, el selector completo siempre es falso.
  - Las operaciones aritméticas deben utilizar la promoción numérica de Java.
- Operador de comparación arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 y arithmetic-expr3:
  - `Age BETWEEN 15 and 19` es equivalente a `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` es equivalente a `age < 15 OR age > 19`.
  - Si cualquiera de las expresiones de una operación BETWEEN es NULL, el valor de la operación es falso. Si cualquiera de las expresiones de una operación NOT BETWEEN es NULL, el valor de la operación es verdadero.
- identificador `[NOT] IN (string-literal1, string-literal2,...)` donde el identificador tiene un valor String o NULL .
  - `Country IN ('UK', 'US', 'France')` es verdadero para `'UK'` y falso para `'Peru'`. Es equivalente a la expresión `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` es false para `'UK'` y true para `'Peru'`. Es equivalente a la expresión `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Si el identificador de una operación IN o NOT IN es NULL, el valor de la operación es desconocido.
- Operador de comparación `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` , donde `identifier` tiene un valor de serie. *valor-patrón* es un literal de tipo serie, donde _ representa un carácter individual cualquiera y % representa una secuencia de caracteres cualquiera (incluida la secuencia vacía). Todos los demás caracteres se representan a sí mismos. El

carácter-escape opcional es un literal de tipo serie formado por un solo carácter que se utiliza para invalidar el significado especial de _ y % en *valor-patrón*. El operador LIKE se debe utilizar sólo para comparar dos valores de tipo serie.

- `phone LIKE '12%3'` es verdadero para 123 y 12993, y falso para 1234.
- `word LIKE 'l_se'` es verdadero para lose y falso para loose.
- `underscored LIKE '\_%' ESCAPE '\'` es verdadero para `_foo` y falso para `bar`.
- `phone NOT LIKE '12%3'` es falso para 123 y 12993 y verdadero para 1234.
- Si el identificador de una operación LIKE o `NOT LIKE` es NULL, el valor de la operación es desconocido.

Nota: El operador LIKE se debe utilizar para comparar dos valores de tipo serie. El valor de `Root.MQMD.CorrelId` es una matriz de bytes de 24 bytes, no una serie de caracteres. El analizador acepta la serie de selector `Root.MQMD.CorrelId LIKE 'ABC%'` como válida sintácticamente, pero se evalúa como falsa. Por lo tanto, cuando compara una matriz de bytes con una serie de caracteres, no se puede utilizar LIKE.

- El operador de comparación `identifier IS NULL` comprueba si existe un valor de campo de cabecera NULL o un valor de propiedad omitido.
- El operador de comparación `identifier IS NOT NULL` comprueba la existencia de un valor de campo de cabecera no nulo o un valor de propiedad.
- Valores nulos

  La evaluación de las expresiones de selector que contienen valores NULL se define mediante la semántica de SQL 92 para valores NULL, en resumen:

  - SQL trata un valor NULL como desconocido.
  - El resultado de una comparación o aritmética con un valor desconocido siempre es un valor desconocido.
  - Los operadores `IS NULL` y `IS NOT NULL` convierten un valor desconocido en los valores TRUE y FALSE.

  Los operadores booleanos utilizan lógica de tres valores (T=TRUE, F=FALSE, U=UNKNOWN)

Tabla 1. Valor del resultado del operador booleano cuando la lógica es A AND B

Operador A	**Operador B**	**Resultado (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

Tabla 2. Valor del resultado del operador booleano cuando la lógica es A OR B

Operador A	**Operador B**	**Resultado (A OR B)**
T	F	T
T	U	T

Tabla 2. Valor del resultado del operador booleano cuando la lógica es A OR B (continuación)		
Operador A	**Operador B**	**Resultado (A OR B)**
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tabla 3. Valor del resultado del operador booleano cuando la lógica es NOT A	
Operador A	**Resultado (NOT A)**
T	F
F	T
U	U

El selector de mensajes siguiente selecciona mensajes con un tipo de mensaje de vehículo, color azul y de peso superior a 2500 libras:

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

Aunque SQL da soporte a la comparación y aritmética decimal fija, los selectores de mensajes no lo hacen. Por ello, los literales numéricos exactos están restringidos a los que no tienen ningún decimal. Esto también es el motivo por el que hay numerales con un decimal como representación alternativa para un valor numérico aproximado.

No se pueden utilizar comentarios de SQL.

Conceptos relacionados

“Propiedades del mensaje” en la página 27
Utilice las propiedades de mensaje para permitir que una aplicación seleccione mensajes para procesar o para recuperar información sobre un mensaje sin acceder a las cabeceras MQMD o MQRFH2. Las propiedades de mensaje también facilitan la comunicación entre las aplicaciones IBM MQ y JMS.

Referencia relacionada

MsgHandle

MQBUFMH - Convertir almacenamiento intermedio en descriptor de contexto de mensaje

Restricciones y reglas de series de selección

Familiarícese con estas reglas acerca de cómo se interpretan las series de selección y las restricciones para evitar problemas potenciales cuando se utilizan selectores.

- La selección de la mensajería de publicación/suscripción se lleva a cabo en el mensaje a medida que la aplicación de publicación envía el mensaje. Consulte la sección Series de selección.
- La equivalencia se prueba utilizando un único carácter de igual. Por ejemplo, a = b es correcto, mientras que a == b es incorrecto.
- Un operador que se utilizan en muchos lenguajes de programación para representar 'no igual a' es !=. Esta representación no es un sinónimo válido para <> ; por ejemplo, a <> b es válido, mientras quea != b no es válido.

- Las comillas simples solo se reconocen si se utiliza el carácter ' (U+0027). Del mismo modo, las comillas dobles, que solo son válidas cuando se utilizan para encerrar series de bytes, deben utilizar el carácter " (U+0022).
- Los símbolos &, &&, | y || no son sinónimos de conjunción/disyunción. Por ejemplo, a `&&` b se debe especificar como a `AND` b.
- Los caracteres comodín `*` y ? no son sinónimos para `%` y `_`.
- Los selectores que contienen expresiones compuestas, tales como `20 < b < 30`, no son válidos. El analizador evalúa los operadores que tienen la misma prioridad de izquierda a derecha. Por lo tanto, el ejemplo se convertiría en `(20 < b) < 30`, lo cual no tiene sentido. En su lugar, la expresión se debe escribir como `(b > 20) AND (b < 30)`.
- Las series de bytes deben estar encerradas entre comillas dobles. Si se utilizan las comillas simples, la serie de bytes se toma como un literal de serie. El número de caracteres, no el número que representan los caracteres, después de `0x` debe ser un múltiplo de dos.
- La palabra clave `IS` no es un sinónimo del carácter de igual. Por lo tanto, las series de selección a `IS 3` y b `IS 'red'` no son válidas. La palabra clave `IS` solo existe para dar soporte a los casos `IS NULL` e `IS NOT NULL`.

Conceptos relacionados

“Comportamiento de la selección” en la página 35
Descripción general del comportamiento de la selección en IBM MQ

Referencia relacionada

Series de selección

Consideraciones sobre UTF-8 y Unicode al utilizar un selector de mensajes

Los caracteres que no están encerrados entre comillas simples y que componen las palabras clave reservadas de una cadena de selección tienen que especificarse en Basic Latin Unicode (que van desde el carácter U+0000 al U+0007F). No es válido utilizar otras representaciones de puntos de código de caracteres alfanuméricos. Por ejemplo, el número 1 ha de expresarse como U+0031 en Unicode, no es válido utilizar el equivalente de dígito de ancho completo U+FF11 o el equivalente en árabe U+0661.

Los nombres de propiedad de mensaje se pueden especificar con cualquier secuencia válida de caracteres Unicode. Los nombres de propiedad de mensaje contenidos en un cadena de selección codificados en UTF-8 serán validados incluso si contienen caracteres multibyte. La validación de UTF-8 multibyte es estricta y hay que asegurarse de que se utilicen secuencias UTF-8 válidas en los nombres de propiedad de mensaje. En los nombres de propiedad de mensaje no se admiten los caracteres más allá de Unicode Basic Multilingual Plane (aquellos que están por encima de U+FFFF), representados en UTF-16 por los puntos de código sustituto (X'D800'a X'DFFF'), o cuatro bytes en UTF-8.

En las comparaciones de igualdad no se realiza ningún procesamiento adicional sobre los nombres de propiedad. Esto significa, por ejemplo, que no se lleva a cabo ninguna composición previa ni descomposición, ni se da ningún significado especial a las ligaduras. Por ejemplo, el carácter de umlaut precompuesto U+00FC no se considera equivalente a U+0075 + U+0308 y la secuencia de caracteres ff no se considera equivalente al Unicode U+FB00 (LATIN SMALL LIGATURE FF)

Los datos de propiedad encerrados entre comillas simples se pueden representar mediante cualquier secuencia de bytes y no se validan.

Selección del contenido de un mensaje

La suscripción puede estar basada en una selección de la carga útil del mensaje, conocida también como filtrado de contenido, pero la decisión acerca de qué mensajes se deben entregar a este tipo de suscripción no puede realizarla directamente IBM MQ, sino que se requiere un proveedor de selección ampliada de mensajes, por ejemplo, IBM Integration Bus, para procesar los mensajes.

Cuando una aplicación publica en una serie de tema, en la que uno o más suscriptores tienen una serie de selecciones sobre el contenido del mensaje, IBM MQ solicita al proveedor de selección ampliada de mensajes que analice la publicación y notifique a IBM MQ si la publicación cumple con el criterio de selección especificado por cada suscriptor con un filtro de contenido.

Si el proveedor de la selección ampliada de mensajes determina que la publicación coincide con la serie de selección del suscriptor, el mensaje continuará entregándose al suscriptor.

Si el proveedor de la selección ampliada de mensajes determina que la publicación no coincide, el mensaje no se entrega al suscriptor. Esto puede hacer que falle la llamada MQPUT o MQPUT1 con el código de razón MQRC_PUBLICATION_FAILURE. Si el proveedor de selección ampliada de mensajes no puede analizar la publicación, se devuelve el código de razón MQRC_CONTENT_ERROR y la llamada MQPUT o MQPUT1 falla.

Si el proveedor de la selección ampliada de mensajes no está disponible o no puede determinar si el suscriptor debe recibir la publicación, se devuelve el código de razón MQRC_SELECTION_NOT_AVAILABLE y la llamada a MQPUT o MQPUT1 falla.

Cuando se crea una suscripción con un filtro de contenido y el proveedor de la selección ampliada de mensajes, la llamada MQSUB falla con el código de razón MQRC_SELECTION_NOT_AVAILABLE. Si se está reanudando una suscripción con un filtro de contenido y el proveedor de selección ampliada de mensajes no está disponible, la llamada MQSUB devuelve el aviso MQRC_SELECTION_NOT_AVAILABLE, pero se permite la reanudación de la suscripción.

Referencia relacionada

Series de selección

Consumo asíncrono de mensajes IBM MQ

El consumo asíncrono utiliza un conjunto de extensiones de interfaz de cola de mensajes (MQI), las llamadas MQI MQCB y MQCTL, que permiten escribir una aplicación MQI para que consuma mensajes de un conjunto de colas. Los mensajes se entregan a la aplicación invocando una 'unidad de código', identificada por la aplicación que pasa el mensaje o por un token que representa el mensaje.

En el más simple de los entornos de aplicación, la unidad de código se define mediante un puntero de función; sin embargo, en otros entornos la unidad de código se puede definir mediante un nombre de programa o módulo.

En el consumo asíncrono de mensajes, se utilizan los términos siguientes:

Consumidor de mensajes
: Programa o función, que se invoca cuando haya un mensaje disponible que responda a los requisitos de una aplicación.

Manejador de sucesos
: Programa o función que se invoca al producirse un suceso asíncrono como, por ejemplo, la desactivación temporal de un gestor de colas.

Devolución de llamada
: Término genérico que se utiliza para hacer referencia a una rutina consumidora mensajes o manejadora de sucesos.

El consumo asíncrono puede simplificar el diseño y la implementación de nuevas aplicaciones, sobre todo aquellas que procesen múltiples colas de entrada o suscripciones. Sin ebargo, si se utiliza más de una cola de entrada y se están procesando mensajes en secuencia de prioridad, esta se respeta de forma independiente dentro de cada cola: Podría ocurrir que se obtuvieran mensajes de baja prioridad de una cola por delante de los mensajes de alta prioridad de otra. El orden de los mensajes procedentes de varias colas no está garantizado. Tenga en cuenta también que si utiliza salidas de API, es posible que tenga que cambiarlas para incluir las llamadas MQCB y MQCTL.

En las ilustraciones siguientes se muestra un ejemplo de cómo se puede utilizar esta función.

Figura 5 en la página 44 muestra una aplicación multihebra que consume mensajes de dos colas. El ejemplo muestra todos los mensajes que se entregan a una sola función.

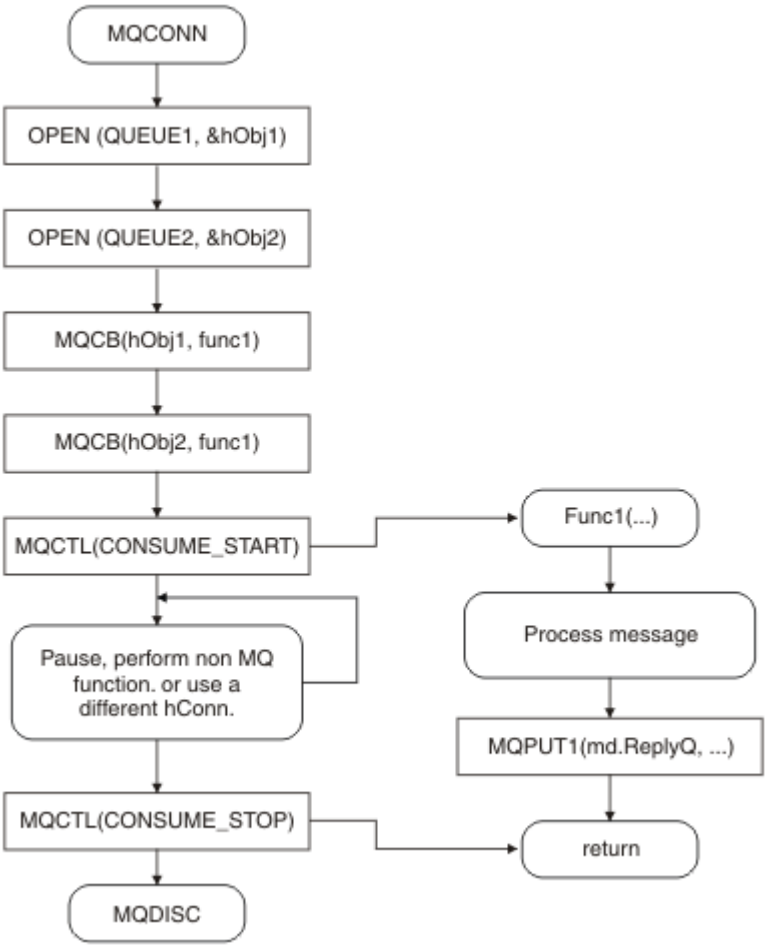


Figura 5. Aplicación controlada por mensajes estándar que consume de dos colas

z/OS En z/OS, la hebra de control principal debe emitir una llamada MQDISC antes de finalizar. Esto permite que las hebras de devolución de llamada terminen y liberen recursos del sistema.

Figura 6 en la página 45 Este flujo de ejemplo muestra una única aplicación con hebras que consume mensajes de dos colas. El ejemplo muestra todos los mensajes que se entregan a una sola función.

La diferencia con el caso asíncrono es que el control no vuelve al emisor de MQCTL hasta que todos los consumidores se hayan desactivado; es decir, un consumidor ha emitido una petición STOP de MQCTL o el gestor de colas se desactiva temporalmente.

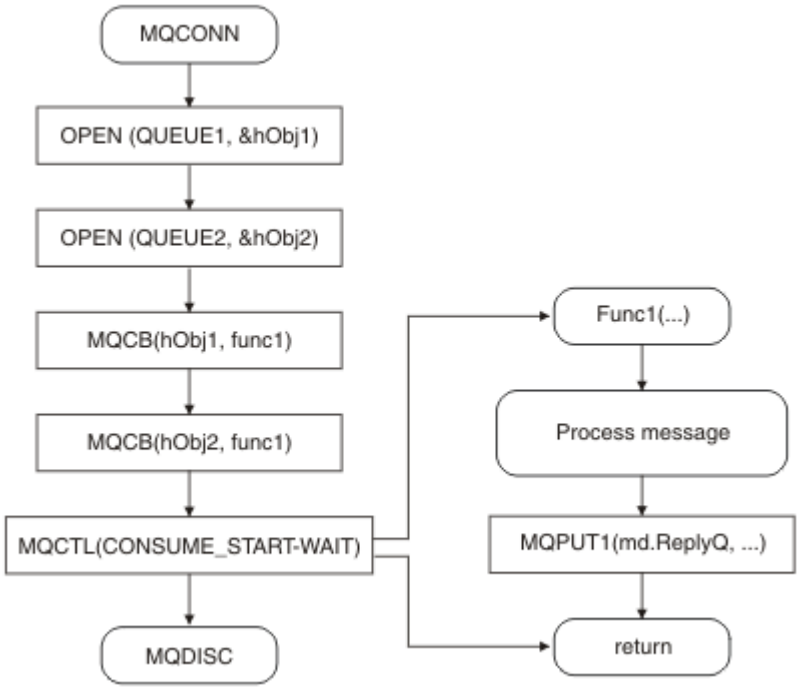


Figura 6. Aplicación controlada por mensajes de hebra única que consume de dos colas

Grupos de mensajes

Los mensajes pueden generarse dentro de grupos, para permitir que se puedan ordenar los mensajes.

Los grupos de mensajes permiten que varios mensajes se marquen como relacionados entre sí, y que se aplique un orden lógico al grupo (consulte "Ordenación lógica y física" en la página 794). En Multiplatforms, la segmentación de mensajes permite dividir los mensajes grandes en segmentos más pequeños. No puede utilizar los mensajes agrupados ni segmentados cuando se transfieren a un tema.

La jerarquía de un grupo es la siguiente:

Grupo
Éste es el nivel más alto de la jerarquía y se identifica mediante un *GroupId*. Consta de uno o varios mensajes que contienen el mismo *GroupId*. Estos mensajes se pueden almacenar en cualquier parte de la cola.

Nota: El término *mensaje* se utiliza aquí para indicar un elemento de una cola que devolvería una MQGET única que no especifique MQGMO_COMPLETE_MSG.

En la Figura 7 en la página 45 se muestra un grupo de mensajes lógicos:

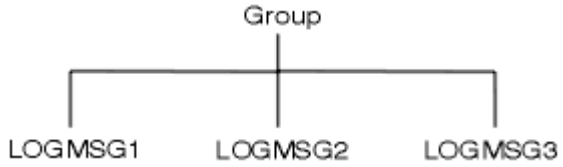


Figura 7. Grupo de mensajes lógicos

Al abrir una cola y especificar MQOO_BIND_ON_GROUP, puede forzar todos los mensajes de un grupo se envían a dicha cola, para que se envíen a la misma instancia de la cola. Para obtener más información sobre la opción BIND_ON_GROUP, consulte la sección Manejo de las afinidades de mensajes.

Mensaje lógico

Los mensajes lógicos de un grupo se identifican mediante los campos *GroupId* y *MsgSeqNumber*. *MsgSeqNumber* empieza en 1 para el primer mensaje que hay en un grupo, y si un mensaje no está en un grupo, el valor del campo es 1.

Utilice los mensajes lógicos de un grupo para:

- Garantizar el orden (si esto no se garantiza bajo las circunstancias en las que se transmite el mensaje).
- Permitir que las aplicaciones agrupen mensajes similares (por ejemplo, aquellos que deba procesar la misma instancia de servidor).

Cada mensaje de un grupo consta de un mensaje físico, a menos que se divida en segmentos. Lógicamente, cada mensaje es un mensaje separado y solo los campos *GroupId* y *MsgSeqNumber* de MQMD deben incluir cualquier relación con otros mensajes del grupo. Los demás campos del MQMD son independientes; algunos pueden ser idénticos para todos los mensajes del grupo, mientras que otros pueden ser diferentes. Por ejemplo, los mensajes de un grupo pueden tener nombres de formatos, CCSID y codificaciones diferentes.

Segmento

Los segmentos se utilizan para manejar mensajes que son demasiado grandes para el gestor de colas o la aplicación que lleva a cabo la transferencia o la obtención (se incluyen los gestores de colas intermedios a través de los cuales pasa el mensaje). Para obtener más información, consulte “Segmentación de mensajes” en la página 813.

Un mensaje individual se divide en mensajes más pequeños, denominados *segmentos*. Un segmento de un mensaje se identifica mediante los campos *GroupId*, *MsgSeqNumber* y *Offset*. El campo *Offset* comienza en cero para el primer segmento que hay dentro de un mensaje.

Cada segmento consiste en un mensaje físico que podría pertenecer a un grupo (en la Figura 8 en la página 46 se muestra un ejemplo de los mensajes que hay un grupo). Lógicamente, un segmento forma parte de único mensaje, por lo tanto, solo los campos *MsgId*, *Offset* y *MsgFlags* de MQMD deben diferir entre los diferentes segmentos del mismo mensaje. Si un segmento no puede llegar, se devuelve el código de razón MQRC_INCOMPLETE_GROUP o MQRC_INCOMPLETE_MSG, según resulte apropiado.

En la Figura 8 en la página 46 se muestra un grupo de mensajes lógicos, algunos de los cuales están segmentados:

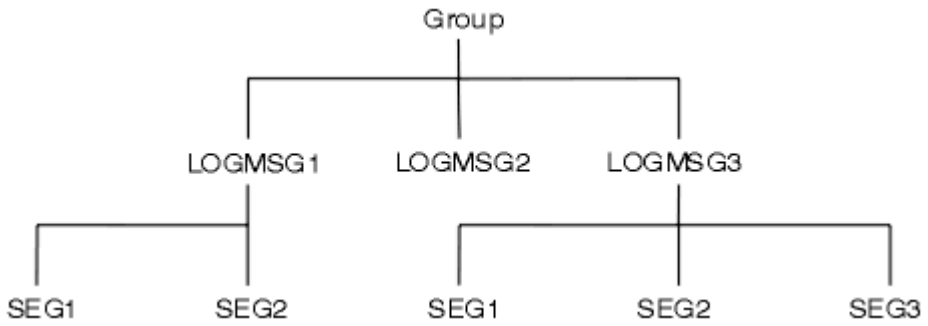


Figura 8. Mensajes segmentados

z/OS En IBM MQ for z/OS no se da soporte a la segmentación.

No puede utilizar los mensajes segmentados ni agrupados con la publicación/suscripción.

Conceptos relacionados

“Segmentación de mensajes” en la página 813
Utilice esta información para obtener información acerca de la segmentación de mensajes. Esta característica no se admite en IBM MQ for z/OS ni mediante aplicaciones que utilizan IBM MQ classes for JMS.

Referencia relacionada

"Ordenación lógica y física" en la página 794
Los mensajes de una cola pueden estar (dentro de cada nivel de prioridad) en orden *físico* o *lógico*.

MQMD - Descriptor de mensaje

Persistencia de los mensajes

Los mensajes persistentes se escriben en archivos de registro y archivos de datos de cola. Si un gestor de colas se reinicia después de un error, recupera estos mensajes persistentes según sea necesario a partir de los datos registrados. Los mensajes que no son persistentes se descartan si se detiene un gestor de colas, tanto si la detención es como resultado de un mandato de operador o debido a un error de alguna parte del sistema.

z/OS Los mensajes no persistentes almacenados en un recurso de acoplamiento (CF) en z/OS son una excepción a esto. Persisten mientras que el CF permanezca disponible.

Cuando crea un mensaje, si inicializa el descriptor de mensaje (MQMD) utilizando los valores predeterminados, la persistencia del mensaje se obtiene del atributo **DefPersistence** de la cola especificada en el mandato MQOPEN. De forma alternativa, puede establecer la persistencia del mensaje utilizando el campo *Persistence* de la estructura MQMD para definir el mensaje como persistente o no persistente.

El rendimiento de la aplicación se ve afectado cuando utiliza mensajes persistentes; la magnitud del efecto dependerá de las características de rendimiento del subsistema de E/S de la máquina y de cómo utilice las opciones de punto de sincronización en cada plataforma:

- Un mensaje persistente, fuera de la unidad de trabajo actual, se graba en disco en cada operación de transferencia y obtención. Consulte "Confirmación y restitución de unidades de trabajo" en la página 874.
- z/OS ALW Para todas las plataformas excepto IBM i, un mensaje persistente dentro de la unidad de trabajo actual sólo se registra cuando se confirma la unidad de trabajo y la unidad de trabajo puede contener muchas operaciones de cola.

Los mensajes no persistentes se pueden utilizar para la mensajería rápida. Consulte Seguridad de los mensajes para obtener más información sobre los mensajes rápidos.

Nota: El escribir mensajes persistentes dentro de una unidad de trabajo y a la vez escribir mensajes persistentes fuera de una unidad de trabajo puede producir problemas de rendimiento, potencialmente graves, en las aplicaciones. Esto es especialmente cierto cuando se utiliza la misma cola de destino para ambas operaciones.

Mensajes que no se pueden entregar

Cuando un gestor de colas no puede poner un mensaje en una cola, tiene varias opciones.

Puede:

- Intentar volver a colocar el mensaje en la cola.
- Solicitar que el mensaje se devuelva al remitente.
- Poner el mensaje en la cola de mensajes no entregados.

Para obtener más información, consulte "Tratamiento de errores en un programa procedimental" en la página 1064.

Mensajes que se restituyen

Cuando se procesan mensajes procedentes de una cola bajo el control de una unidad de trabajo, la unidad de trabajo puede estar formada por uno o más mensajes. Si se realiza una restitución, los mensajes que se hayan recuperado de la cola se restablecen en esta y se pueden procesar de nuevo en otra unidad de trabajo. Si el proceso de un determinado mensaje está provocando el problema, se

restituye de nuevo la unidad de trabajo. Esto puede provocar un bucle de proceso. Los mensajes que se hayan colocado en una cola se eliminan de la misma.

Una aplicación puede detectar los mensajes que están atrapados en un bucle, probando el campo *BackoutCount* de MQMD. La aplicación puede corregir la situación o puede emitir un aviso a un operador.

El recuento de restituciones siempre sobrevive a los reinicios del gestor de colas. Los cambios efectuados en el atributo **HardenGetBackout** se ignoran.

Para las colas compartidas, el recuento de restituciones siempre sobrevive a los reinicios del gestor de colas. Para las demás configuraciones de z/OS, para asegurarse de que el recuento de restituciones de las colas privadas sobrevive a los reinicios del gestor de colas, establezca el atributo *HardenGetBackout* en MQQA_BACKOUT_HARDENED; de lo contrario, si el gestor de colas tiene que reiniciarse, no mantiene un recuento de restituciones exacto para cada mensaje. Si se establece el atributo de este modo, se añade el coste del proceso adicional.

Para obtener más información sobre cómo confirmar y restituir mensajes, consulte "Confirmación y restitución de unidades de trabajo" en la página 874.

Cola de respuesta y gestor de colas

Hay ocasiones en que recibirá mensajes en respuesta a un mensaje que ha enviado:

- Un mensaje de respuesta en respuesta a un mensaje de solicitud
- Un mensaje de informe sobre una caducidad o suceso inesperado
- Un mensaje de informe sobre una COA (Confirmación de llegada) o un suceso de COD (Confirmación de entrega)
- Un mensaje de informe sobre un suceso de PAN (Notificación de acción positiva) o suceso de NAN (Notificación de acción negativa).

Mediante la estructura MQMD, especifique el nombre de la cola a la que desee que se envíen los mensajes de respuesta y de informe en el campo *ReplyToQ*. Especifique el nombre del gestor de colas que es el propietario de la cola de respuesta en el campo *ReplyToQMgr*.

Si deja el campo *ReplyToQMgr* en blanco, el gestor de colas establece el contenido de los campos siguientes en el descriptor de mensaje en la cola:

ReplyToQ
Si *ReplyToQ* es una definición local de una cola remota, el campo *ReplyToQ* se establece en el nombre de la cola remota; en caso contrario, este campo no cambia.

ReplyToQMgr
Si *ReplyToQ* es una definición local de una cola remota, el campo *ReplyToQMgr* se establece en el nombre del gestor de colas que sea propietario de la cola remota; en caso contrario, el campo *ReplyToQMgr* se establece en el nombre del gestor de colas al que esté conectada la aplicación.

Nota: Puede solicitar que un gestor de colas realice más de un intento para entregar un mensaje, y puede solicitar que el mensaje se descarte si falla. Si el mensaje, después de no poder entregarse, no se debe descartar, el gestor de colas remoto coloca el mensaje en la cola de mensajes no entregados (consulte "Utilización de la cola de mensajes no entregados" en la página 1068).

Contexto de mensaje

La información del *contexto de mensaje* permite a la aplicación que recupera el mensaje averiguar quién ha originado el mensaje.

Es posible que la aplicación que efectúa la recuperación desee:

- Comprobar que la aplicación emisora tenga el nivel correcto de autorización
- Realizar algunas funciones de contabilidad para cargar a la aplicación emisora los trabajos que tenga que realizar

- Mantener un seguimiento de auditoría de todos los mensajes con los que haya trabajado

Cuando utilice la llamada MQPUT o MQPUT1 para colocar un mensaje en una cola, puede especificar que el gestor de colas añada alguna información de contexto predeterminada en el descriptor de mensaje. Las aplicaciones que tengan el nivel apropiado de autorización pueden añadir información de contexto adicional. Para obtener más información sobre cómo especificar la información de contexto, consulte "Control de la información de contexto de mensaje" en la página 779.

El gestor de colas utiliza el contexto de usuario cuando genera los siguientes tipos de mensaje de informe:

- Confirmar al entregar
- Caducidad

Cuando se generan estos mensajes de informe, se comprueban en el contexto de usuario las autorizaciones +put y +passid en el destino del informe. Si el contexto de usuario no tiene autorización suficiente, el mensaje de informe se coloca en la cola de mensajes no entregados, si se ha definido alguna. Si no hay ninguna cola de mensajes no entregados, el mensaje de informe se descarta.

Toda la información de contexto se almacena en los campos de contexto del descriptor de mensaje. El tipo de información incluye la información de contexto de identidad, origen y usuario.

Contexto de identidad

La información del *contexto de identidad* identifica al usuario de la aplicación que ha colocado primero el mensaje en una cola. Las aplicaciones debidamente autorizadas pueden establecer los campos siguientes:

- El gestor de colas rellena el campo *UserIdentifier* con un nombre que identifica al usuario; el modo en que el gestor de colas puede hacerlo depende del entorno en el que se ejecuta la aplicación.
- El gestor de colas rellena el campo *AccountingToken* con una señal o un número que se determina en la aplicación que coloca el mensaje.
- Las aplicaciones puede utilizar el campo *ApplIdentityData* para cualquier información adicional que deseen incluir sobre el usuario (por ejemplo, una contraseña cifrada).

Se almacena un identificador de seguridad de sistemas (SID) de Windows en el campo *AccountingToken* cuando se crea un mensaje en IBM MQ for Windows. El SID se puede utilizar para complementar el campo *UserIdentifier* y para establecer las credenciales de un usuario.

Para obtener información sobre cómo el gestor de colas llena los campos *UserIdentifier* y *AccountingToken*, consulte las descripciones de estos campos en UserIdentifier y AccountingToken.

Las aplicaciones que pasan los mensajes de un gestor de colas a otro también deben pasar la información de contexto de identidad, para que las demás aplicaciones conozcan la identidad del originador del mensaje.

Contexto de origen

La información del *contexto de origen* describe la aplicación que ha colocado el mensaje en la cola en la que está almacenado actualmente. El descriptor de mensaje contiene los campos siguientes para poder obtener información del contexto de origen:

- *PutApplType* define el tipo de aplicación que ha colocado el mensaje (por ejemplo, una transacción CICS).
- *PutApplName* define el nombre de la aplicación que ha colocado el mensaje (por ejemplo, el nombre de un trabajo o una transacción).
- *PutDate* define la fecha en la que se ha colocado el mensaje en la cola.
- *PutTime* define la hora en la que se ha colocado el mensaje en la cola.

- *ApplOriginData* define cualquier información adicional que una aplicación desee incluir sobre el origen del mensaje. Por ejemplo, lo podrían establecer aplicaciones debidamente autorizadas para indicar si los datos de identidad son fiables.

La información de contexto de origen la proporciona normalmente el gestor de colas. La hora media de Greenwich (GMT) se utiliza para los campos *PutDate* y *PutTime*. Consulte las descripciones de estos campos en PutDate y PutTime.

Una aplicación con suficiente autorización puede proporcionar su propio contexto. Esto permite conservar la información de contabilidad cuando un único usuario tiene un ID de usuario distinto en cada uno de los sistemas que procesan un mensaje que ellos mismos han originado.

Objetos de IBM MQ

Esta información proporciona detalles sobre objetos IBM MQ que incluyen: gestores de colas, grupos de compartición de colas, colas, objetos de tema administrativo, listas de nombres, definiciones de proceso, objetos de información de autenticación, canales, clases de almacenamiento, escuchas y servicios.

Los gestores de colas definen las propiedades (que se conocen como atributos) de estos objetos. Los valores de estos atributos afectan a la forma en la que IBM MQ procesa estos objetos. Desde las aplicaciones, se utiliza la Interfaz de cola de mensajes (MQI) para controlar estos objetos. Los objetos se identifican mediante un *descriptor de objetos* (MQOD) cuando se direccionan desde un programa.

Cuando se utilizan mandatos de IBM MQ para definir, alterar o suprimir objetos, por ejemplo, el gestor de colas comprueba que tiene el nivel de autorización necesario para realizar estas operaciones. Del mismo modo, cuando una aplicación utiliza la llamada MQOPEN para abrir un objeto, el gestor de colas comprueba si la aplicación dispone del nivel de autorización necesario antes de permitir el acceso a dicho objeto. Las comprobaciones se realizan en el nombre del objeto que se abre.

Conceptos relacionados

"Control de la información de contexto de mensaje" en la página 779
Cuando utilice la llamada MQPUT o MQPUT1 para colocar un mensaje en una cola, puede especificar que el gestor de colas añada alguna información de contexto predeterminada en el descriptor de mensaje. Las aplicaciones que tengan el nivel apropiado de autorización pueden añadir información de contexto adicional. Puede utilizar el campo de opciones de la estructura MQPMO para controlar la información de contexto.

Referencia relacionada

"Opciones de MQOPEN relacionadas con el contexto del mensaje" en la página 769
Si desea poder asociar información de contexto a un mensaje cuando lo coloca en una cola, debe utilizar una de las opciones de contexto de mensaje al abrir la cola.

Windows Preparación y ejecución de aplicaciones Microsoft Transaction Server

Para preparar una aplicación MTS para que se ejecute como una aplicación IBM MQ MQI client, siga estas instrucciones según corresponda para el entorno.

Para obtener información general sobre cómo desarrollar aplicaciones Microsoft Transaction Server (MTS) que acceden a recursos IBM MQ, consulte la sección sobre MTS en el IBM MQ Help Center.

Para preparar una aplicación MTS para que se ejecute como una aplicación IBM MQ MQI client, realice una de las acciones siguientes para cada componente de la aplicación:

- Si el componente utiliza los enlaces del lenguaje C en la MQI, siga las instrucciones de "Preparación de programas C en Windows" en la página 1042, pero enlace el componente con la biblioteca mqicxa.lib en vez de mqic.lib.
- Si el componente utiliza las clases C++ de IBM MQ, siga las instrucciones de "Compilación de programas C++ en Windows" en la página 567, pero enlace el componente a la biblioteca imqx23vn.lib, en lugar de imqc23vn.lib.

- Si el componente utiliza los enlaces de lenguaje Visual Basic para la MQI, siga las instrucciones que aparecen en "Preparación de programas Visual Basic en Windows" en la página 1045, pero cuando defina el proyecto Visual Basic, escriba MqType=3 en el campo **Argumentos de compilación condicional**,

Consideraciones acerca del diseño de las aplicaciones IBM MQ

Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

Durante el diseño de una aplicación IBM MQ tenga en cuenta las siguientes preguntas y opciones:

Tipo de aplicación

¿Cuál es la finalidad de la aplicación? Vea los enlaces siguientes para obtener información sobre los diferentes tipos de aplicaciones que puede desarrollar:

- Servidor
- Cliente
- Publicación/suscripción
- Servicios Web
- Salidas de usuario, Salidas de API y servicios instalables

Además también puede escribir sus propias aplicaciones para automatizar la administración de IBM MQ. Para obtener más información, consulte las secciones La interfaz de administración de IBM MQ (MQAI) y Automatización de tareas de administración.

Lenguaje de programación

IBM MQ da soporte a una serie de lenguajes de programación distintos para la escritura de aplicaciones. Si desea más información, consulte "Desarrollo de aplicaciones para IBM MQ" en la página 5.

Aplicaciones para más de una plataforma

¿La aplicación se ejecutará en más de una plataforma? ¿Dispone de una estrategia para pasar a una plataforma diferente de la que utiliza hoy? Si la respuesta a cualquiera de estas preguntas es sí, asegúrese de codificar los programas para la independencia de plataforma.

Por ejemplo, si está utilizando C, codifique en el estándar C de ANSI. Utilice una función de biblioteca C estándar en lugar de una función específica de plataforma equivalente incluso si la función específica de plataforma es más rápida o más eficiente. La excepción es si la eficacia del código es importante cuando debe codificar en ambos casos utilizando #ifdef. Por ejemplo:

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

Tipos de colas

¿Desea crear una cola cada vez que necesite una o bien desea utilizar colas que ya se han configurado? ¿Desea suprimir una cola cuando haya terminado de utilizarla o bien va a utilizarla de nuevo? ¿Desea utilizar colas alias para la independencia de aplicación? Para ver qué tipos de colas están soportados, consulte la sección Colas.

z/OS **Utilización de colas compartidas, grupos de compartición de colas y clústeres de grupos de compartición de colas (solo IBM MQ for z/OS)**

Es posible que desee beneficiarse del aumento de la disponibilidad, la escalabilidad y el equilibrio de carga de trabajo que son posibles cuando se utilizan colas compartidas con grupos de compartición de colas. Para obtener más información, consulte Colas compartidas y grupos de compartición de colas.

También es posible que desee realizar una estimación de los flujos máximo y promedio de mensajes y considerar el uso de clústeres de grupos de compartición de colas para repartir la carga de trabajo. Para obtener más información, consulte Colas compartidas y grupos de compartición de colas.

Utilización de los clústeres del gestor de colas
Es posible que desee aprovechar la administración del sistema simplificado y una mayor disponibilidad, escalabilidad y equilibrio de carga de trabajo que son posibles cuando se utilizan clústeres.

Tipos de mensajes
Es posible que desee utilizar datagramas para los mensajes simples, pero mensajes de solicitud (para el que se esperan respuestas) para otras situaciones. Tal vez desee asignar prioridades diferentes a algunos de los mensajes. Para obtener más información acerca del diseño de mensajes, consulte "Técnicas de diseño para mensajes" en la página 60.

Utilización de la publicación/suscripción o la mensajería punto a punto
Mediante la mensajería de publicación/suscripción, una aplicación emisora envía la información que desea compartir en un mensaje IBM MQ a un destino estándar gestionado mediante la publicación/suscripción de IBM MQ y permite que IBM MQ maneje la distribución de dicha información. La aplicación de destino no tiene que saber nada sobre la fuente de información que recibe, sólo registra un interés en uno o más temas y recibe esa información cuando esté disponible. Para obtener más información sobre la mensajería de publicación/suscripción, consulte Mensajería de publicación/suscripción.

Utilizando la mensajería punto a punto, una aplicación emisora envía un mensaje a una cola específica, desde donde se sabe que una aplicación de recepción va a recuperarlo. Una aplicación receptora obtiene mensajes de una cola específica y actúa sobre su contenido. A menudo, una aplicación funcionará como un remitente y un destinatario, enviando una consulta a otra aplicación y recibiendo una respuesta.

Controlar sus programas IBM MQ
Es posible que desee iniciar algunos programas automáticamente o hacer que los programas esperen hasta que llegue un mensaje determinado a una cola (utilizando la característica IBM MQ *desencadenamiento* , consulte "Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886). De forma alternativa, es posible que desee iniciar otra instancia de una aplicación cuando los mensajes de una cola no se procesen lo suficientemente rápido (utilizando la característica IBM MQ *sucesos de instrumentación* tal como se describe en Sucesos de instrumentación).

Ejecutar su aplicación en un cliente de IBM MQ
En el entorno de cliente se da soporte completo a MQI y prácticamente cualquier aplicación IBM MQ escrita en un lenguaje de procedimiento se puede volver a enlazar para ejecutarla en un IBM MQ MQI client. Enlace la aplicación del IBM MQ MQI client con la biblioteca MQIC, en lugar de con la biblioteca MQI. z/OS No se da soporte a Get(signal) en z/OS.

Nota: Una aplicación que se ejecuta en un cliente de IBM MQ se puede conectar a más de un gestor de colas al mismo tiempo, o puede utilizar un nombre de gestor de colas con un asterisco (*) en una cola MQCONN o MQCONNX. Cambie la aplicación si desea enlazar a las bibliotecas del gestor de colas en vez de enlazar a las bibliotecas de cliente porque esta función no estará disponible.

Consulte "Ejecución de aplicaciones en el entorno de IBM MQ MQI client" en la página 944 para obtener más información.

Rendimiento de la aplicación
Las decisiones sobre diseño pueden afectar al rendimiento de la aplicación, para obtener sugerencias sobre cómo mejorar el rendimiento de las aplicaciones IBM MQ, consulte la sección "Consideraciones sobre el diseño de aplicaciones y el rendimiento" en la página 62 IBM i y la sección "Consideraciones de diseño y rendimiento para aplicaciones de IBM i" en la página 65.

Técnicas avanzadas de IBM MQ
Para aplicaciones más avanzadas, puede ser conveniente utilizar algunas técnicas avanzadas de IBM MQ tales como la correlación de respuestas y la generación y envío de información de contexto de

IBM MQ. Para obtener más información, consulte “Técnicas de diseño de aplicaciones avanzadas” en la página 64.

Protección de los datos y mantenimiento de la integridad

Puede utilizar la información de contexto que se pasa con un mensaje para comprobar que el mensaje se ha enviado desde un origen aceptable. Puede utilizar los recursos de puntos de sincronismo que proporciona IBM MQ, o su sistema operativo, para asegurarse de que sus datos se mantienen coherentes con otros recursos (consulte la sección “Confirmación y restitución de unidades de trabajo” en la página 874 para obtener más detalles). Puede utilizar la característica de *persistencia* de los mensajes de IBM MQ para asegurar la entrega de mensajes importantes.

Probar aplicaciones IBM MQ

El entorno de desarrollo de aplicaciones para programas IBM MQ no es diferente del de cualquier otra aplicación, de modo que puede utilizar las mismas herramientas de desarrollo, así como los recursos de rastreo de IBM MQ.

Cuando pruebe las aplicaciones CICS con IBM MQ for z/OS, puede utilizar CICS Execution Diagnostic Facility (CEDF). CEDF atrapa la entrada y salida de cada llamada MQI, así como las llamadas a todos los servicios CICS. Asimismo, en el entorno CICS, puede escribir un programa de salida entre varias API para proporcionar información de diagnóstico antes y después de cada llamada MQI. Para obtener información sobre cómo conseguirlo consulte el apartado “Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911.

Al realizar pruebas en las aplicaciones IBM i, puede utilizar el depurador estándar. Para iniciarlo, utilice el mandato STRDBG.

Manejo de excepciones y errores

Debe considerar cómo procesar los mensajes que no se pueden entregar, y cómo resolver situaciones de error sobre las que el gestor de colas le informado. Para algunos informes, debe establecer opciones de informe en MQPUT.

Conceptos relacionados

Visión general técnica de IBM MQ

“Consideraciones de diseño y rendimiento para aplicaciones de z/OS” en la página 67
El diseño de aplicaciones es uno de los factores más importantes que afectan al rendimiento. Utilice este tema para conocer algunos de los factores de diseño que participan en el rendimiento.

“Desarrollo de aplicaciones para IBM MQ” en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

“Conceptos de desarrollo de aplicaciones” en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

“Desarrollo de una aplicación procedimental para encolamientos” en la página 740
Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

“Desarrollo de aplicaciones procedimentales cliente” en la página 936
Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

“Desarrollo de aplicaciones .NET” en la página 571
IBM MQ classes for .NET permitir que las aplicaciones .NET se conecten a IBM MQ como un IBM MQ MQI client o que se conecten directamente a un servidor IBM MQ .

“Desarrollo de aplicaciones C++” en la página 543
IBM MQ proporciona clases C++ equivalentes a los objetos IBM MQ y algunas clases equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

“Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85
IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

“Utilización de IBM MQ classes for Java” en la página 356
Utilice IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

Especificación del nombre de aplicación en los lenguajes de programación admitidos

Antes de IBM MQ 9.2.0, ya podría especificar un nombre de aplicación en las aplicaciones cliente de Java o JMS. A partir de IBM MQ 9.2.0, esta característica se amplía a otros lenguajes de programación en IBM MQ for Multiplatforms.

Cómo se utiliza el nombre de aplicación

El nombre de la aplicación es la salida de:

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE(HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- Rastreo de actividad de la aplicación

El nombre de la aplicación también se utiliza al configurar el rastreo de la actividad de la aplicación. El nombre de aplicación predeterminado para aplicaciones que no son deJava es el nombre truncado del ejecutable, excepto en Windows y IBM i.

Windows En Windows, el nombre predeterminado es el nombre ejecutable completo, truncado a 28 caracteres a la izquierda.

IBM i En IBM i, el nombre predeterminado es el nombre del trabajo.

Para las aplicaciones Java, se trata del nombre de clase prefijado por el nombre del paquete truncado a la izquierda en 28 caracteres.

Para obtener más información, consulte PutApplName.

A partir de IBM MQ 9.2.0, las aplicaciones en IBM MQ for Multiplatforms pueden establecer sus nombres de aplicación de forma administrativa o utilizando distintos métodos de programación. Esto permite a las aplicaciones proporcionar una nombre más significativo, independiente de la plataforma, cuando se configura el rastreo de la actividad de la aplicación o cuando se genera la salida de varios mandatos **runmqsc**.

A partir de IBM MQ 9.2.0, puede reequilibrar las aplicaciones en un clúster uniforme. Para conseguir esto se utilizan nombres de aplicación significativos.

Caracteres soportados

Consulte “Caracteres recomendados para el nombre de aplicación” en la página 55 si desea más información sobre cómo especificar el nombre de aplicación.

Lenguajes de programación

Consulte “Conexiones de lenguaje de programación” en la página 57 para obtener más información sobre cómo las aplicaciones, que se resuelven en las bibliotecas de IBM MQ en y otros lenguajes de programación, pueden proporcionar el nombre de la aplicación.

Aplicaciones .NET gestionadas

Consulte “Aplicaciones .NET gestionadas” en la página 58 si desea más información sobre cómo las aplicaciones .NET gestionadas pueden proporcionar el nombre de la aplicación.

Aplicaciones XMS

Consulte “XMSAplicaciones” en la página 59 si desea más información sobre cómo las aplicaciones XMS pueden proporcionar el nombre de la aplicación.

Aplicaciones de enlaces Java y JMS

ALW

Consulte “Aplicaciones de enlaces Java y JMS” en la página 59 si desea más información sobre cómo las aplicaciones Java y JMS pueden proporcionar el nombre de la aplicación.

Conceptos relacionados

Rastreo de actividad de la aplicación

Acerca de los clústeres uniformes

Referencia relacionada

MQCNO

MQCNO en IBM i

Utilización del nombre de aplicación en lenguajes de programación admitidos

Utilice esta información para obtener información sobre cómo se selecciona el nombre de aplicación en los distintos lenguajes que IBM MQ admite.

Caracteres recomendados para el nombre de aplicación

Los nombres de aplicación deben estar en el juego de caracteres proporcionado por el atributo **CodedCharSetId** del campo del gestor de colas. Para obtener más información sobre este atributo, consulte Atributos para el gestor de colas.

Sin embargo, si la aplicación se está ejecutando como un IBM MQ MQI client, el nombre de la aplicación debe estar en el juego de caracteres y en la codificación del cliente.

Para garantizar una transición fluida del nombre de aplicación entre gestores de colas y permitir la supervisión de recursos de aplicación a través de los temas de supervisión de recursos, los nombres de aplicación sólo deben contener caracteres imprimibles de un solo byte.

Notas:

- También debería evitar el uso de la barra inclinada y los caracteres ampersand en los nombres de aplicación.
- V 9.3.0 Debe evitar el uso del carácter ampersand en los nombres de aplicación. No se producirán las métricas STATAPP del tema del sistema para los nombres de aplicación que contienen un ampersand.

Esto limita el nombre a:

- Caracteres alfanuméricos: `A-Z`, `a-zy 0-9`

Nota: No debe utilizar los caracteres a - z en minúsculas en los nombres de aplicación en sistemas que utilizan EBCDIC Katakana.

- El carácter de espacio
- Caracteres imprimibles que son invariables en EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- V 9.3.0 El carácter/. Al suscribirse al rastreo de actividad o a las métricas de tema del sistema STATAPP para una aplicación cuyo nombre contiene una barra inclinada, debe sustituir los caracteres de barra inclinada por un carácter ampersand. Por ejemplo, para recibir métricas STATAPP para una aplicación denominada "DEPT1/APPS/STOCKQUOTE" debe suscribirse a la serie de tema "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE". Las aplicaciones de ejemplo amqsact y amqsrua convertirán automáticamente los caracteres de barra inclinada a ampersand al crear sus suscripciones.

Cómo se establecen los caracteres

En la tabla siguiente se resumen los medios por los que el nombre de aplicación se elige en los distintos lenguajes que IBM MQ admite. El medio por el que se elige el nombre está en orden de prioridad, el más alto en primer lugar.

	Enlaces de C y cliente	**Enlaces de Java y cliente**	**Enlaces de JMS y cliente**	**Cliente .NET gestionado**	**Enlaces .NET no gestionados y cliente**	**Cliente XMS gestionado**	**Enlaces y cliente .XMS no gestionados**
Alteración temporal de la propiedad de conexión		Alteración temporal de la propiedad de conexión de Java		Alteración temporal de la propiedad de conexión de .NET	Alteración temporal de la propiedad de conexión de .NET		
Propiedad alterada temporalmente		Propiedad alterada temporalmente de Java		Propiedad alterada temporalmente de .NET	Propiedad alterada temporalmente de .NET		
MQEnvironment		Java Entorno MQEntorno		.NET MQEnvironment	.NET MQEnvironment		
Propiedad de fábrica de conexiones			Propiedad de fábrica de conexiones			Propiedad de fábrica de conexiones	Propiedad de fábrica de conexiones

	Enlaces de C y cliente	Enlaces de Java y cliente	Enlaces de JMS y cliente	Cliente .NET gestionado	Enlaces .NET no gestionados y cliente	Cliente XMS gestionado	Enlaces y cliente .XMS no gestionados
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	Opciones de conexión						
Variable de entorno	Variables de entorno				Variables de entorno		Variables de entorno
mqclient.ini (Aplicable a las conexiones de cliente solamente)	Conexiones de cliente				Conexiones de cliente		Conexiones de cliente
Nombre de clase Java		Nombre de clase Java	Nombre de clase Java				
Nombre predeterminado	Nombre predeterminado			Nombre predeterminado de .NET	Nombre predeterminado de .NET	Nombre predeterminado de .NET	Nombre predeterminado de .NET

Nota: La columna de enlaces C y cliente se aplica a los lenguajes de programación siguiente también:

- COBOL
- Ensamblador
- Visual Basic
- RPG

Conexiones de lenguaje de programación

Las aplicaciones que se resuelven en las bibliotecas IBM MQ en C, y otros lenguajes de programación, pueden proporcionar el nombre de la aplicación de las formas siguientes.

Los métodos de conexión se enumeran en orden de prioridad, empezando por el más alto.

Multi **Opciones de conexión**

- ALW MQCNO

  Nota: z/OS Al conectarse a un gestor de colas de IBM MQ for z/OS , sólo puede establecer el nombre de aplicación utilizando conexiones en modalidad de cliente, o utilizando aplicaciones IBM MQ classes for JMS o IBM MQ classes for Java .

- IBM i MQCNO en IBM i

ALW Variables de entorno

Si todavía no ha seleccionado un nombre de aplicación, puede utilizar la variable de entorno **MQAPPLNAME** para identificar la conexión con el gestor de colas. Por ejemplo:

```
export MQAPPLNAME=ExampleApplName
```

Tenga en cuenta que solo se utilizan los primeros 28 caracteres y estos caracteres no deben ser todos espacios en blanco o nulos.

Nota: El atributo sólo se aplica a los lenguajes de programación soportados, a las conexiones .NETno gestionadas y a las conexiones XMS no gestionadas.

ALW Archivo de configuración de cliente

Si todavía no ha seleccionado un nombre de aplicación y la conexión es una conexión de cliente, puede especificar la información siguiente en el archivo de configuración de cliente (por ejemplo, `mqclient.ini`) para identificar la conexión con el gestor de colas.

```
Connection:
    ApplName=ExampleApplName
```

Notas:

1. Solo se utilizan los primeros 28 caracteres y estos caracteres no deben ser todos espacios en blanco o nulos.
2. El atributo sólo se aplica a las conexiones de cliente en los lenguajes de programación soportados, las conexiones .NETno gestionadas y las conexiones XMS no gestionadas.

Para obtener más información, consulte el archivo de configuración de IBM MQ MQI client , `mqclient.ini`.

Nombre predeterminado

Si todavía no ha elegido el nombre de aplicación, se sigue utilizando el nombre predeterminado, que contiene tanto la vía de acceso como el nombre del ejecutable como se muestra en el sistema operativo. Para obtener más información, consulte PutApplName.

Aplicaciones .NET gestionadas

Las aplicaciones .NET gestionadas pueden proporcionar el nombre de la aplicación de las formas siguientes.

Los métodos de conexión se enumeran en orden de prioridad, empezando por el más alto.

Alteración temporal de la propiedad de conexión

Puede proporcionar un archivo de alteración temporal de detalles de conexión a las aplicaciones de la forma siguiente:

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

El archivo especificado por `overrideConnectionDetailsFile` contiene una lista de propiedades precedidas por `mqj`. Las aplicaciones deben definir la propiedad **`mqj.APPNAME`** donde el valor de la propiedad **`mqj.APPNAME`** especifica el nombre utilizado para identificar la conexión al gestor de colas.

Solo se utilizan los primeros 28 caracteres del nombre. Por ejemplo:

```
mqj.APPNAME=ExampleApplName
```

Propiedad alterada temporalmente

Se ha definido una constante **MQC.APPNAME_PROPERTY** con el valor *APPNAME*. Ahora puede pasar esta propiedad al constructor **MQQueueManager**, utilizando los primeros 28 caracteres solo del nombre. Por ejemplo:

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

Para obtener más información, consulte “Operaciones gestionadas o no gestionadas en .NET” en la página 671.

MQEnvironment

La propiedad *AppName* se añade a la clase **MQEnvironment**y sólo se utilizan los primeros 28 caracteres. Por ejemplo:

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

Nombre predeterminado

Si no ha proporcionado el nombre de la aplicación por alguno de los medios descritos en el texto anterior, el nombre de la aplicación se establece automáticamente como el nombre ejecutable (y como gran parte de la vía de acceso que se ajustará).

XMSAplicaciones

Los métodos de conexión se enumeran en orden de prioridad, empezando por el más alto.

Propiedad de fábrica de conexiones

Las aplicaciones XMS pueden proporcionar el nombre de aplicación en la fábrica de conexiones utilizando la propiedad **XMSC.WMQ_APPLICATIONNAME** ("*XMSC_WMQ_APPNAME*") de forma similar a JMS. Puede especificar un máximo de 28 caracteres.

Para obtener más información, consulte “XMS .NET Creación de objetos administrados” en la página 679 y “Propiedades de un mensaje XMS” en la página 687.

JMSAdmin

En las herramientas administrativas, la propiedad se conoce como "**APPLICATIONNAME**" o "**APPNAME**" para abreviar.

Aplicaciones de enlaces Java y JMS

Los métodos de conexión se enumeran en orden de prioridad, empezando por el más alto.

ALW Las aplicaciones cliente Java y JMS ya pueden especificar un nombre de aplicación, y esto se ha ampliado en IBM MQ for Multiplatforms para las aplicaciones de enlaces, utilizando el campo **ApplName** de MQCNO.

Alteración temporal de la propiedad de conexión

La propiedad **Application name** se ha añadido a la lista de propiedades de conexión que puede alterar temporalmente. Para obtener más información, consulte Utilización de la alteración temporal de la propiedad de conexión IBM MQ.

Atención: Las propiedades de conexión y la forma de utilizar el archivo de alteración temporal de propiedad de conexión para ambos, IBM MQ classes for Java y .NET.

Propiedad alterada temporalmente

Se ha definido una constante **MQC.APPNAME_PROPERTY** con el valor *APPNAME*. Ahora puede pasar esta propiedad al constructor **MQQueueManager**, utilizando los primeros 28 caracteres solo del nombre. Para obtener más información, consulte Utilización de la alteración temporal de la propiedad de conexión en IBM MQ classes for Java.

MQEnvironment

La propiedad *AppName* se añade a la clase **MQEnvironment** y sólo se utilizan los primeros 28 caracteres.

Para obtener más información, consulte “Configuración del entorno de IBM MQ para IBM MQ classes for Java” en la página 384.

Nombre de clase Java

Si no ha proporcionado el nombre de aplicación por alguno de los medios del texto anterior, el nombre de aplicación se deriva del nombre de clase principal.

Para obtener más información, consulte “Configuración del entorno de IBM MQ para IBM MQ classes for Java” en la página 384.

Atención: IBM i En IBM i no es posible consultar el nombre de clase principal, por lo que se utiliza IBM MQ client para Java en su lugar.

Conceptos relacionados

“Configuración del entorno de IBM MQ para IBM MQ classes for Java” en la página 384
Para que una aplicación se pueda conectar a un gestor de colas en la modalidad de cliente, la aplicación debe especificar el nombre de canal, el nombre de host y el número de puerto.

Referencia relacionada

MQCNO
MQCNO en IBM i

Técnicas de diseño para mensajes

Consideraciones para ayudarle a diseñar mensajes, incluidas consideraciones para selectores y propiedades de mensaje.

Cosas para tener en cuenta en la etapa de diseño

Se crea un mensaje para colocarlo en una cola mediante una llamada MQI. Como entrada a la llamada, se proporciona información de control en un *descriptor de mensaje* (MQMD) y los datos que desea enviar a otro programa. Pero, en la etapa de diseño, hay que tener en cuenta lo siguiente, ya que afecta a la forma en que se crean los mensajes:

Tipo de mensaje que se usa

¿Va a diseñar una aplicación sencilla en la que se puede enviar un mensaje y después no hacer nada? ¿O solicita una respuesta a una pregunta? Si hace una pregunta, en el descriptor de mensaje puede incluir el nombre de la cola en la que desea recibir la respuesta.

¿Desea que los mensajes de solicitud y de respuesta sean síncronos? Esto implica establecer un periodo de tiempo de espera a la respuesta a su solicitud y, si no se recibe una respuesta en el transcurso de ese periodo, se considerará un error.

¿Prefiere trabajar de forma asíncrona, de modo que los procesos no tengan que depender de si se producen sucesos específicos tales como señales de temporización habituales?

También hay que tener en cuenta si todos los mensajes se hallan dentro de una unidad de trabajo.

Asignación de diferentes prioridades a un mensaje

Se puede asignar un valor de prioridad a cada mensaje y definir la cola para que mantenga sus mensajes por orden de prioridad. Si lo hace, cuando otro programa recupere un mensaje de la cola, siempre recuperará el mensaje que tenga la prioridad más alta. Si la cola no mantiene sus mensajes en orden de prioridad, los programas recuperarán los mensajes de la cola en el orden en el que se añadieron a la cola.

Los programas también pueden seleccionar un mensaje utilizando el identificador asignado por el gestor de colas cuando dicho mensaje se coloca en la cola. De forma alternativa, puede generar sus propios identificadores para cada uno de los mensajes.

Efecto de un reinicio del gestor de colas sobre los mensajes

El gestor de colas conserva todos los mensajes permanentes, recuperándolos cuando es necesario de los archivos de registro IBM MQ, cuando se reinicia. Los mensajes no persistentes y las colas dinámicas temporales no se conservan. Los mensajes cuyo descarte no se desee tendrán que definirse como persistentes cuando se crean. Al escribir una aplicación para IBM MQ for Windows o IBM MQ en sistemas AIX and Linux, asegúrese de que sabe cómo se ha configurado el sistema con respecto a la asignación de archivos de registro para reducir el riesgo de diseñar una aplicación que ejecutará los límites del archivo de registro.

z/OS Puesto que los mensajes en colas compartidas (solo disponibles en IBM MQ for z/OS) se mantienen en el recurso de acoplamiento (CF), los mensajes no persistentes se conservan entre los reinicios de un gestor de colas, siempre que el CF esté disponible. Si el CF falla, se perderán los mensajes no persistentes.

Cómo dar información sobre uno mismo al destinatario de un mensaje

Generalmente, el gestor de mensajes establece el ID de usuario, pero las aplicaciones que tengan la autorización pertinente también pueden establecer este campo para que se pueda incluir el propio ID de usuario y otra información que el programa receptor puede usar en auditorías y cuestiones de seguridad.

Cantidad de colas de recepción

Multi Si fuera necesario colocar un mensaje en varias colas, se podría colocar en un tema o en una lista de distribución.

z/OS Si fuera necesario colocar un mensaje en varias colas, se podría colocar en un tema.

Selectores y propiedades de mensaje

Los mensajes pueden tener asociados metadatos junto a la carga útil del mensaje principal. Estas propiedades de mensaje pueden ser útiles para suministrar de datos adicionales.

Es importante tener en cuenta dos aspectos de estos datos adicionales:

- Las propiedades no están sujetas a la protección Advanced Message Security (AMS). Si desea utilizar AMS para proteger los datos, colóquelos en la carga útil y no en las propiedades del mensaje.
- Las propiedades se pueden utilizar para seleccionar mensajes.

Es importante tener en cuenta que el uso de selectores rompe la convención de mensajes estándar 'primero en entrar, primero en salir'. Puesto que el gestor de colas está optimizado para esta carga de trabajo, no se recomienda usar selectores complejos por motivos de rendimiento. El gestor de colas no almacena los índices de las propiedades del mensaje; por tanto, la búsqueda de un mensaje tiene que ser lineal. Cuanto más profunda sea la cola, más complejo será el selector y una menor probabilidad de que el selector encuentre una coincidencia supondrá una penalización del rendimiento.

Si se requiere una selección compleja, se recomienda filtrar los mensajes utilizando cualquier aplicación o motor de procesamiento como, por ejemplo, IBM Integration Bus, a destinos diferentes. De forma alternativa, el uso de una jerarquía de temas podría ser útil.

Nota: IBM MQ classes for Java no admite el uso de selectores, si desea utilizar selectores, esto se debe realizar a través de la API JMS.

Consideraciones sobre el diseño de aplicaciones y el rendimiento

Un diseño deficiente puede afectar al rendimiento de diversas maneras. Esto puede ser difícil de detectar ya que el programa puede dar la impresión de que funciona bien pero al mismo tiempo afectar al rendimiento de otras tareas. En este tema se explican varios problemas específicos a la realización de llamadas IBM MQ por parte de los programas.

A continuación se muestran algunas ideas para ayudarle a diseñar aplicaciones eficientes:

- Diseñe su aplicación de manera que el proceso vaya en paralelo con el tiempo de reflexión del usuario:
  - Muestre un panel y permita que el usuario empiece a escribir mientras la aplicación aún se está inicializando.
  - Obtenga los datos que necesite en paralelo desde distintos servidores.
- Mantenga las conexiones y colas abiertas si va a reutilizarlas en lugar de abrir y cerrar, conectar y desconectar de forma repetida.
- No obstante, una aplicación de servidor que solo pone un mensaje debe utilizar MQPUT1.
- Los gestores de colas están optimizados para mensajes de un tamaño entre 4 KB y 100 KB. Los mensajes muy grandes son ineficientes; probablemente sea mejor enviar 100 mensajes de 1 MB cada uno que un único mensaje de 100 MB. Los mensajes muy pequeños también son ineficientes. El gestor de colas realiza la misma cantidad de trabajo para un mensaje de un único byte que para un mensaje de 4 KB.
- Mantenga sus mensajes dentro de una unidad de trabajo para que se puedan confirmar o restituir de forma simultánea.
- Utilice la opción nonpersistent (no persistente) para mensajes que no tengan que ser recuperables.
- Si necesita enviar un mensaje a una serie de colas de destino, plantéese utilizar una lista de distribución.

Efecto de la longitud del mensaje

La cantidad de datos que contiene el mensaje puede afectar al rendimiento de la aplicación que procesa el mensaje. Para mejorar el rendimiento desde su aplicación, envíe solo los datos esenciales en el mensaje. Por ejemplo, en una solicitud de cargo en una cuenta bancaria, la única información que es preciso transmitir del cliente a la aplicación servidor es el número de cuenta y el importe del cargo.

Efecto de la persistencia de los mensajes

Los mensajes persistentes se anotan en el archivo de anotaciones. La anotación de los mensajes reduce el rendimiento de la aplicación, por lo que solamente debe utilizar mensajes persistentes para los datos esenciales. Si los datos de un mensaje se pueden descartar cuando el gestor de colas se detiene o finaliza con error, utilice un mensaje no persistente.

z/OS Las operaciones MQPUT y MQGET para mensajes persistentes se bloquearán cuando no haya suficiente espacio de registro de recuperación para registrar las operaciones. Dicha condición se indica en el registro de trabajos del gestor de colas mediante los mensajes CSQJ110E y CSQJ111A. Asegúrese de que los procesos de supervisión estén en su lugar para que se gestionen y eviten dichas condiciones.

Búsqueda de un mensaje determinado

La llamada MQGET suele recuperar el primer mensaje de una cola. Si utiliza los identificadores de mensaje y de correlación (*MsgId* y *CorrelId*) en el descriptor de mensajes para especificar un mensaje determinado, el gestor de colas deberá buscar en la cola hasta encontrar ese mensaje. Utilizar la llamada MQGET de este modo afecta al rendimiento de la aplicación.

Colas que contienen mensajes de distintas longitudes

Si la aplicación no puede utilizar mensajes de longitud fija, puede aumentar y reducir el tamaño de los almacenamientos intermedios de forma dinámica para que se ajuste al tamaño de mensaje típico. Si la aplicación emite una llamada MQGET que no se realiza correctamente porque el almacenamiento intermedio es demasiado pequeño, se devuelven el tamaño de los datos del mensaje. Añada código a su aplicación para que el almacenamiento intermedio se redimensione en consonancia y se vuelva a emitir la llamada MQGET.

Nota: Si no establece de forma explícita el atributo **MaxMsgLength**, será de 4 MB de forma predeterminada, que podría ser muy ineficiente si se utiliza para influir en el tamaño del almacenamiento intermedio de aplicación.

Frecuencia de los puntos de sincronización

Los programas que emiten grandes cantidades de llamadas MQPUT o MQGET dentro del punto de sincronización sin confirmarlas pueden provocar problemas de rendimiento. Las colas afectadas pueden llenarse de mensajes inaccesibles en ese momento, mientras otras tareas pueden estar esperando obtener esos mensajes. Esto repercute en el almacenamiento y en las hebras asociadas a tareas que intentan obtener mensajes.

Utilización de la llamada MQPUT1

La llamada MQPUT1 solo se debe usar si hay un único mensaje que transferir a una cola. Si desea transferir más de un mensaje, utilice la llamada MQOPEN seguida de una serie de llamadas MQPUT y de una sola llamada MQCLOSE.

Número de hebras usadas

Windows Para IBM MQ for Windows, es posible que una aplicación necesite una gran cantidad de hebras. Cada proceso de gestor de colas tiene asignado un número máximo permitido de hebras de aplicación.

Las aplicaciones pueden utilizar demasiadas hebras. Vea si la aplicación tiene en cuenta esta posibilidad y si toma medidas para poner freno a este tipo de problema o informar sobre él.

Colocar mensajes persistentes dentro de un punto de sincronización

Los mensajes persistentes deben colocarse y obtenerse bajo un punto de sincronización. Esto se debe a que, al obtener un mensaje permanente fuera de un punto de sincronización, si la obtención falla, la aplicación no tiene un modo de saber si el mensaje se ha obtenido de la cola o no, ni tampoco si, en caso de que se haya obtenido el mensaje, también se haya perdido. Al obtener mensajes persistentes bajo un punto de sincronización, si algo falla, se retrotrae la transacción y el mensaje persistente no se pierde, ya que sigue estando en la cola.

De forma similar, al poner mensajes persistentes, póngalos también bajo un punto de sincronización. Otro motivo para poner y obtener mensajes persistentes bajo un punto de sincronización es que el código del mensaje persistente en IBM MQ está altamente optimizado para el punto de sincronización. Así, poner y obtener mensajes persistentes bajo un punto de sincronización es más rápido que poner y obtener mensajes persistentes fuera de un punto de sincronización.

Si la aplicación coloca mensajes persistentes fuera del punto de sincronización, el gestor de colas comprueba si puede crear un punto de sincronización implícito en nombre de la aplicación. Si el gestor de colas puede hacerlo, incluye la colocación dentro de ese punto de sincronización y la confirma automáticamente. Consulte “Punto de sincronismo implícito en Multiplatforms” en la página 883 para obtener una descripción más detallada.

No obstante, es más rápido poner y obtener mensajes no persistentes fuera de un punto de sincronización, ya que el código no persistente en IBM MQ está optimizado para estar fuera del punto de sincronización. Poner y obtener mensajes persistentes se hace a velocidad de disco, ya que el mensaje

persistente se mantiene en disco. No obstante, poner y obtener mensajes no persistentes se realiza a velocidades de CPU, ya que no hay ninguna escritura en disco implicada, incluso cuando se utiliza un punto de sincronización.

Si una aplicación obtiene mensajes y no conoce de antemano si son persistentes o no, se puede utilizar la opción MQGMO_SYNCPOINT_IF_PERSISTENT de GMO.

Técnicas de diseño de aplicaciones avanzadas

Al diseñar aplicaciones más avanzadas, hay algunas técnicas que puede que le interese considerar como, por ejemplo, esperar mensajes, correlacionar respuestas, definir y utilizar información de contexto, iniciar aplicaciones automáticamente, generar informes y eliminar afinidades de mensajes al utilizar una agrupación en clúster.

Para una aplicación simple de IBM MQ, tendrá que decidir qué objetos IBM MQ utilizar en la aplicación y qué tipos de mensaje desea utilizar. En el caso de una aplicación más avanzada, puede que le interese usar algunas de las técnicas presentadas en las secciones siguientes.

Espera de mensajes

Un programa que da servicio a una cola puede esperar mensajes:

- Esperando a que llegue un mensaje o a que venza un intervalo de tiempo determinado (consulte "Espera de mensajes" en la página 818).
- z/OS Solo en IBM MQ for z/OS, definiendo una señal para que se informe al programa cuando llega un mensaje. Para obtener más información, consulte "Uso de señales" en la página 818.
- Estableciendo una salida de devolución de llamada (callback) que se accione cuando llegue un mensaje; consulte "Consumo asíncrono de mensajes IBM MQ" en la página 43.
- Efectuando llamadas periódicas a la cola para ver si ha llegado un mensaje (*sondeo*). Por lo general, esto no es aconsejable porque puede penalizar el rendimiento.

Correlación de respuestas

En las aplicaciones IBM MQ, cuando un programa recibe un mensaje que le solicita que realice algún trabajo, el programa suele enviar uno o más mensajes de respuesta al solicitante.

Para ayudar al solicitante a asociar estas respuestas con su solicitud original, una aplicación puede establecer el campo *identificador de correlación* en el descriptor de cada mensaje. Luego los programas copian el identificador del mensaje de solicitud en el identificador de correlación de sus mensajes de respuesta.

Definición y uso de información de contexto

La *Información de contexto* se usa para asociar mensajes con el usuario que los genera y para identificar la aplicación que ha generado un mensaje. Esta información es útil a efectos de seguridad, contabilidad, auditoría y determinación de problemas.

Al crearse un mensaje, se puede especificar una opción que solicite que el gestor de colas asocie información de contexto predeterminada al mensaje.

Para obtener más información sobre la definición y el uso de información de contexto, consulte "Contexto de mensaje" en la página 48.

Inicio automático de programas IBM MQ

Utilice IBM MQ *triggering* para iniciar un programa automáticamente cuando lleguen mensajes a una cola.

Se pueden definir condiciones de desencadenante en una cola para que un programa empiece a procesar dicha cola:

- Cada vez que un mensaje llega a la cola.
- Cuando el primer mensaje llega a la cola.
- Cuando el número de mensajes de la cola alcanza un número predefinido.

Para más información sobre el desencadenamiento, consulte “Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886. Los desencadenantes no son más que una forma de iniciar un programa automáticamente. Por ejemplo, se podría iniciar un programa de forma automática con la ayuda de un temporizador usando recursos que no sean de IBM MQ.

En Multiplatforms, IBM MQ puede definir objetos de servicio para iniciar programas IBM MQ cuando se inicia el gestor de colas; consulte Objetos de servicio.

Generación de informes de IBM MQ

Se puede solicitar los informes siguientes en una aplicación:

- Informes de excepciones.
- Informes de caducidad.
- Informes de confirmación de llegada (COA).
- Informes de Confirmación de entrega (COD).
- Informes de notificación de acción positiva (PAN).
- Informes de notificación de acción negativa (NAN).

Estos se describen en “Mensajes de informe” en la página 21.

Clústeres y afinidades de mensajes

Antes de empezar a utilizar clústeres con varias definiciones para la misma cola, examine las aplicaciones para ver si hay alguna que requiera un intercambio de mensajes relacionados.

En un clúster, un mensaje se puede direccionar a cualquier gestor de colas que aloje una instancia de la correspondiente cola. Por lo tanto, esto puede alterar la lógica de las aplicaciones con afinidades de mensajes.

Por ejemplo, se podrían tener dos aplicaciones basadas en una serie de mensajes que fluyen entre ellas en forma de preguntas y respuestas. Podría ser importante que todas las preguntas se envíen al mismo gestor de colas y que todas las respuestas se devuelvan al otro gestor de colas. En esta situación, es importante que la rutina de gestión de carga de trabajo no envíe los mensajes a un gestor de colas simplemente porque este aloje una instancia de la correspondiente cola.

En la medida de lo posible, elimine las afinidades. La eliminación de las afinidades de mensajes mejora la disponibilidad y la escalabilidad de las aplicaciones.

Para obtener más información, consulte Manejo de las afinidades de mensajes.

Consideraciones de diseño y rendimiento para aplicaciones de IBM i

Utilice esta información para comprender cómo el diseño de aplicaciones, las hebras y el almacenamiento pueden afectar al rendimiento.

Esta información se divide en dos secciones:

- “Consideraciones sobre el diseño de aplicaciones” en la página 65
- “Problemas específicos de rendimiento” en la página 67

Consideraciones sobre el diseño de aplicaciones

Un diseño deficiente puede afectar al rendimiento de diversas maneras. Estos problemas pueden ser difíciles de detectar porque puede parecer que el programa tiene buen rendimiento, pero puede afectar

al rendimiento de otras tareas. En las siguientes secciones se explican diversos problemas específicos de los programas que efectúan llamadas de IBM MQ for IBM i.

Para obtener más información sobre el diseño de aplicaciones, consulte "Consideraciones acerca del diseño de las aplicaciones IBM MQ" en la página 51.

Efecto de la longitud del mensaje
Aunque IBM MQ for IBM i permite que los mensajes tengan más de 100 MB de datos, la cantidad de datos en un mensaje afecta al rendimiento de la aplicación que procesa el mensaje. Para obtener el máximo rendimiento de una aplicación, envíe solamente los datos esenciales en un mensaje; por ejemplo, en una petición de cargar en una cuenta bancaria, la única información que podría ser preciso pasar del cliente a la aplicación de servidor es el número de cuenta y el importe del cargo.

Efecto de la persistencia de los mensajes
Los mensajes persistentes se registran por diario. El registro por diario de los mensajes penaliza el rendimiento de la aplicación, por lo que solamente hay que utilizar mensajes permanentes para los datos esenciales. Si los datos de un mensaje se pueden descartar cuando el gestor de colas se detiene o finaliza con error, utilice un mensaje no persistente.

Búsqueda de un mensaje determinado
La llamada MQGET suele recuperar el primer mensaje de una cola. Si utiliza los identificadores de mensaje y de correlación (*MsgId* y *CorrelId*) en el descriptor de mensaje para especificar un mensaje determinado, el gestor de colas tiene que buscar en la cola hasta encontrar ese mensaje. El hecho de utilizar de esta manera la llamada MQGET penaliza el rendimiento de la aplicación.

Colas que contienen mensajes de distintas longitudes
Si los mensajes de una cola tienen longitudes diferentes, para determinar el tamaño de un mensaje, la aplicación puede utilizar la llamada MQGET con el campo *BufferLength* establecido en cero para que, aunque la llamada falle, devuelva el tamaño de los datos del mensaje. La aplicación puede repetir a continuación la llamada, especificando el identificador del mensaje que ha obtenido en la primera llamada y un búfer del tamaño correcto. Sin embargo, si hay otras aplicaciones que utilizan la misma cola, el rendimiento de la aplicación puede verse penalizado porque la segunda llamada MQGET invierte tiempo en buscar un mensaje que otra aplicación ya ha recuperado en el intervalo de tiempo transcurrido entre ambas llamadas.

Si la aplicación no puede utilizar mensajes de longitud fija, otra solución a este problema sería utilizar la llamada MQINQ para buscar el tamaño máximo de mensaje que la cola puede aceptar y utilizar dicho valor en la llamada MQGET. El tamaño máximo de los mensajes de una cola se almacena en el atributo **MaxMsgLen** de la cola. Este método podría utilizar grandes cantidades de almacenamiento, sin embargo, porque el valor de este atributo de colas puede ser el máximo permitido por IBM MQ for IBM i, que puede ser mayor que 2 GB.

Frecuencia de los puntos de sincronización
Los programas que emiten numerosas llamadas MQPUT dentro de un punto de sincronización, sin comprometerlos, pueden provocar problemas de rendimiento. Las colas afectadas pueden llenarse de mensajes que no se pueden utilizar en ese momento, mientras otras tareas pueden estar a la espera de obtener esos mensajes. Este problema repercute en el almacenamiento y en las hebras asociadas a tareas que intentan obtener mensajes.

Utilización de la llamada MQPUT1
La llamada MQPUT1 solo se debe usar si hay un único mensaje que transferir a una cola. Si desea transferir más de un mensaje, utilice la llamada MQOPEN seguida de una serie de llamadas MQPUT y de una sola llamada MQCLOSE.

Número de hebras usadas
Una aplicación puede necesitar muchas hebras. A cada proceso de gestor de colas se le asigna un número máximo de hebras permitidas. Si algunas aplicaciones son problemáticas, podría deberse a que, por diseño, usan demasiadas hebras. Vea si la aplicación tiene en cuenta esta posibilidad y si toma medidas para poner freno a este tipo de problema o informar sobre él. El número máximo de hebras que permite IBM i es de 4,095. No obstante, el valor predeterminado es 64. IBM MQ pone a disposición de sus procesos hasta 63 hebras.

Problemas específicos de rendimiento

En esta sección se explican los problemas de almacenamiento y rendimiento deficientes.

Problemas de almacenamiento

Si recibe el mensaje del sistema `CPF0907. Serious storage condition may exist.`, es posible que esté llenando el espacio asociado con los gestores de colas de IBM MQ for IBM i .

¿La aplicación o IBM MQ for IBM i funcionan con lentitud?

Si su aplicación ejecuta con lentitud, podría ser síntoma de que ha entrado en bucle o está esperando un recurso que no está disponible. Esta ejecución lenta también puede deberse a un problema de rendimiento. Puede que el sistema esté funcionando al límite de su capacidad. Es probable que este tipo de problema se agrave en las horas punta de carga del sistema, que suelen ser a media mañana y a media tarde. (Si la red abarca más de un huso horario, podría parecer que el pico de carga del sistema se produce a otras horas del día).

Si averigua que la degradación del rendimiento no depende de la carga del sistema, sino que a veces se produce cuando la carga del sistema es mínima, la causa puede residir en una aplicación mal diseñada. Esto puede manifestarse como un problema que solo se produce al acceder a determinadas colas.

QTOTJOB y QADLTOTJ son valores del sistema que merece la pena investigar.

Los síntomas siguientes podrían indicar que IBM MQ for IBM i está funcionando lentamente:

- El sistema tarda en responder a los comandos MQSC.
- Si repetidas visualizaciones de la profundidad de la cola indican que la cola se está procesando lentamente para una aplicación con la que sería de esperar una gran cantidad de actividad de cola.
- ¿Se está ejecutando el rastreo de IBM MQ?

Linux Consideraciones acerca del diseño de las aplicaciones Linux on Power Systems - Little Endian

Puesto que Linux on Power Systems - Little Endian solo da soporte a aplicaciones de 64-bits, no hay ningún soporte proporcionado en IBM MQ para aplicaciones de 32-bits.

Conceptos relacionados

“Consideraciones acerca del diseño de las aplicaciones IBM MQ” en la página 51

Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

z/OS Consideraciones de diseño y rendimiento para aplicaciones de z/OS

El diseño de aplicaciones es uno de los factores más importantes que afectan al rendimiento. Utilice este tema para conocer algunos de los factores de diseño que participan en el rendimiento.

Un diseño deficiente puede afectar al rendimiento de diversas maneras. Estos problemas pueden ser difíciles de detectar porque puede parecer que el programa tiene buen rendimiento, pero puede afectar al rendimiento de otras tareas. En las secciones siguientes se muestran varios problemas específicos de los programas que hacen llamadas MQI.

Para obtener más información sobre el diseño de aplicaciones, consulte “Consideraciones acerca del diseño de las aplicaciones IBM MQ” en la página 51.

Efecto de la longitud del mensaje

Aunque IBM MQ for z/OS permite que los mensajes tengan más de 100 MB de datos, la cantidad de datos en un mensaje afecta al rendimiento de la aplicación que procesa el mensaje. Para mejorar el rendimiento desde su aplicación, envíe solo los datos esenciales en el mensaje. Por ejemplo, en una solicitud de cargo en una cuenta bancaria, la única información que es preciso transmitir del cliente a la aplicación servidor es el número de cuenta y el importe del cargo.

Efecto de la persistencia de los mensajes

Se registran los mensajes persistentes. La anotación de los mensajes reduce el rendimiento de la aplicación, por lo que solamente debe utilizar mensajes persistentes para los datos esenciales. Si los datos de un mensaje se pueden descartar cuando el gestor de colas se detiene o finaliza con error, utilice un mensaje no persistente.

Los datos de los mensajes persistentes se escriben en los almacenamientos intermedios de registro. Estos almacenamientos intermedios se escriben en los conjuntos de datos de registro cuando:

- Se produce una confirmación
- Un mensaje ha quedado (o se ha colocado) fuera del punto de sincronización
- Los almacenamientos intermedios de WRTHRSH están llenos

El proceso de muchos mensajes en una unidad de trabajo puede provocar menos entrada/salida que si los mensajes se procesaran cada uno en una unidad de trabajo, o fuera del punto de sincronización.

Búsqueda de un mensaje determinado

La llamada MQGET suele recuperar el primer mensaje de una cola. Si utiliza los identificadores de mensaje y de correlación (**MsgId** y **CorrelId**) que hay en el descriptor de mensaje para especificar un mensaje determinado, el gestor de colas busca en la cola hasta que encuentra ese mensaje. El uso de MQGET de este modo afecta al rendimiento de la aplicación porque, para encontrar un mensaje determinado, es posible que IBM MQ tenga que explorar toda la cola.

Puede utilizar el atributo de cola **IndexType** para especificar que desea que el gestor de colas mantenga un índice que se puede utilizar para aumentar la velocidad de las operaciones de MQGET en la cola. Sin embargo, hay una pequeña reducción de rendimiento para el mantenimiento de un índice, por lo que sólo lo generaría si necesita usarlo. Puede elegir crear un índice de identificadores de mensaje o de identificadores de correlación, o bien puede optar por no crear un índice para las colas en las que los mensajes se recuperan secuencialmente. Intente que haya muchos valores de clave diferentes, no muchas con el mismo valor. Por ejemplo, Saldo1, Saldo2 y Saldo3, en lugar de Saldo tres veces. Para las colas compartidas, debe tener el **IndexType** correcto. Para obtener detalles sobre el atributo de cola **IndexType**, consulte IndexType.

Para evitar que afecte al tiempo de reinicio del gestor de cola usando colas indexadas, utilice el parámetro QINDXBLD(NOWAIT) en la macro CSQ6SYSP. Esto permite que se complete el reinicio del gestor de colas sin esperar a que se complete la creación del índice de cola.

Para obtener una descripción completa del atributo **IndexType**, y otros atributos de objeto, consulte Atributos de objetos.

Colas que contienen mensajes de distintas longitudes

Obtenga un mensaje, utilizando un tamaño de almacenamiento intermedio que coincida con el tamaño esperado del mensaje. Si recibe el código de retorno que indica que el mensaje es demasiado largo, obtenga un almacenamiento intermedio más grande. Cuando el get falla de esta forma, la longitud de datos devuelta es el tamaño de los datos de mensaje no convertidos. Si especifica MQGMO_CONVERT en la llamada MQGET, y los datos se expanden durante la conversión, aún podría no encajar en el almacenamiento intermedio, en cuyo caso deberá aumentar más su tamaño.

Si emite la MQGET con una longitud de almacenamiento intermedio cero, devuelve el tamaño del mensaje y la aplicación podrá así obtener un almacenamiento intermedio de dicho tamaño y volver a emitir la operación get. Si tiene varias aplicaciones que procesan la cola, es posible que otra aplicación ya haya procesado el mensaje cuando la aplicación original haya vuelto a emitir la operación get. Si ocasionalmente tiene mensajes grandes, es posible que tenga que obtener un almacenamiento intermedio grande sólo para estos mensajes, y liberarlo después de que se haya procesado el mensaje.

Esto debería contribuir a reducir los problemas de almacenamiento virtual si todas las aplicaciones tienen almacenamientos intermedios grandes.

Si la aplicación no puede utilizar mensajes de longitud fija, otra solución a este problema es utilizar la llamada MQINQ para buscar el tamaño máximo de mensaje que la cola puede aceptar y utilizar dicho valor en la llamada MQGET. El tamaño máximo de los mensajes de una cola se almacena en el atributo **MaxMsgL** de la cola. Este método podría utilizar grandes cantidades de almacenamiento, sin embargo, porque el valor de **MaxMsgL** podría ser de hasta 100 MB, el máximo permitido por IBM MQ for z/OS.

Nota: Puede bajar el parámetro **MaxMsgL** después de que se hayan colocado mensajes grandes en la cola. Por ejemplo, puede poner un mensaje de 100 MB y, a continuación, establecer **MaxMsgL** en 50 bytes. Esto significa que todavía es posible obtener mensajes más grandes de lo que espera la aplicación.

Frecuencia de los puntos de sincronización

Los programas que emiten muchas llamadas de MQPUT dentro de un punto de sincronización, sin confirmarlas, pueden provocar problemas de rendimiento. Las colas afectadas pueden llenarse de mensajes que no se pueden utilizar en ese momento, mientras otras tareas pueden estar a la espera de obtener esos mensajes. Esto repercute en el almacenamiento y en las hebras asociadas a tareas que intentan obtener mensajes.

Como regla, si tiene varias aplicaciones que procesan una cola, normalmente se obtiene el mejor rendimiento cuando hay

- 100 mensajes cortos (menos de 1 KB), o
- Un mensaje para mensajes más grandes (100 KB)

para cada punto de sincronización. Si sólo hay una aplicación procesando la cola, debe tener más mensajes para cada unidad de trabajo.

Puede limitar el número de mensajes que una tarea puede obtener o colocar dentro de una única unidad de recuperación con el atributo del gestor de colas **MAXUMSGS**. Para obtener información sobre este atributo, consulte el mandato **ALTER QMGR** en los mandatos MQSC.

Ventajas de la llamada MQPUT1

La llamada MQPUT1 solo se debe usar si hay un único mensaje que transferir a una cola. Si desea transferir más de un mensaje, utilice la llamada MQOPEN, seguida de una serie de llamadas de MQPUT y una única llamada MQCLOSE.

¿Cuántos mensajes puede contener un gestor de colas?

Colas locales

El número de mensajes locales que un gestor de colas puede contener es básicamente el tamaño de los conjuntos de páginas. Puede tener un máximo de 100 conjuntos de páginas (aunque se recomienda el conjunto de páginas 0 y el conjunto de páginas 1 para los objetos y colas relacionados con el sistema). Puede utilizar un conjunto de páginas con formato ampliado y aumentar la capacidad de un conjunto de páginas.

Colas compartidas

La capacidad de las colas compartidas depende del tamaño del recurso de acoplamiento (CF). IBM MQ utiliza estructuras de lista de CF en las que las unidades de almacenamiento fundamentales son entradas y elementos. Cada mensaje se almacena como 1 entrada y varios elementos que contienen el MQMD asociado y otros datos de mensaje. El número de elementos consumidos por un solo mensaje depende del tamaño del mensaje y, para CFLEVEL (5), de las reglas de descarga en vigor en el momento de MQPUT. Se necesitan menos elementos cuando los datos de mensaje se descargan

en Db2 o en SMDS. El acceso a los datos de mensajes es más lento cuando el mensaje se ha descargado. Consulte Performance Supportpac MP1H para obtener una comparación más completa de la sobrecarga de rendimiento y de CPU asociada con la descarga de mensajes.

¿Qué afecta al rendimiento?

El rendimiento puede ser la velocidad a la que se pueden procesar los mensajes, pero también la cantidad de CPU que se necesita por mensaje.

¿Qué afecta a la velocidad en la que se pueden procesar los mensajes?

Para los mensajes persistentes, el mayor impacto es la velocidad de los conjuntos de datos de registro. La velocidad de los conjuntos de datos de registro depende del DASD en el que estén. Por lo tanto, se debe tener cuidado de poner el conjunto de datos de registro en volúmenes de bajo uso para reducir la contención. La escritura en bandas de los registros de MQ mejora el rendimiento del registro cuando hay varias páginas grabadas por E/S. La conexión de fibra Z High Performance (zHPF) también tiene un rendimiento significativo en el tiempo de respuesta de E/S cuando el subsistema de E/S está ocupado.

Cuando hay una solicitud para obtener y poner un mensaje, el acceso a la cola se bloquea durante la solicitud para conservar la integridad de la cola. A efectos de planificación, tenga en cuenta que la cola está bloqueada para toda la solicitud. Así que si el tiempo para una puesta es de 100 microsegundos, y tiene más de 10.000 peticiones un segundo, puede que se produzcan retrasos. Puede mejorar en la práctica, pero es una buena regla general. Puede utilizar distintas colas para mejorar el rendimiento.

Las razones posibles para esto pueden ser:

- utilizar una cola de respuestas común que utilicen todas las transacciones de CICS
- a cada transacción de CICS se le proporciona una respuesta exclusiva a la cola
- una respuesta a una cola para la región de CICS y todas las transacciones de la región de CICS utilizan esta cola.

La respuesta depende del número de solicitudes por segundo y del tiempo de respuesta de las solicitudes.

Si hay que leer los mensajes de un conjunto de páginas, será más lento en comparación con cuando los mensajes están en la agrupación de almacenamiento intermedio. Si tiene más mensajes de los que caben en una agrupación de almacenamiento intermedio, entonces se desbordarán al disco. Así que tiene que asegurarse de que la agrupación de almacenamiento intermedio sea lo suficientemente grande para los mensajes de corta vida. Si tiene mensajes que procesa muchas horas más tarde, es probable que se desborden al disco, por lo que debería esperar que estos mensajes sean más lentos que si estuvieran en la agrupación de almacenamiento intermedio.

En el caso de una cola compartida, la velocidad de los mensajes depende de la velocidad del recurso de acoplamiento. Es probable que un CF dentro del procesador físico sea más rápido que un CF externo. El tiempo de respuesta de CF depende de lo ocupado que esté el CF. Por ejemplo, en los sistemas Hursley, cuando el CF tenía un 17 % de ocupación, el tiempo de respuesta era de 14 microsegundos. Cuando el CF estaba ocupado en un 95 %, el tiempo de respuesta era de 45 microsegundos.

Si las solicitudes de MQ utilizan mucha CPU, esto puede afectar a la velocidad en la que se procesan los mensajes. Porque si la partición lógica (LPAR) está restringida para la CPU, las aplicaciones se retrasarán a la espera de CPU.

Cuánta CPU por mensaje

En general, los mensajes más grandes utilizan más CPU, por lo es bueno evitar en lo posible los mensajes grandes (x MB).

Al obtener mensajes específicos de las colas, la cola se debe indexar para que el gestor de colas pueda ir directamente al mensaje (evitando así un posible exploración completa de la cola). Si no se

indexa la cola, se explora desde el principio para buscar el mensaje. Si hay 1000 mensajes en la cola, es posible que tenga que explorarlos todos. Eso supone un gran uso innecesario de la CPU.

Los canales que utilizan TLS tienen un coste adicional debido al cifrado del mensaje.

En MQ v7 puede seleccionar mensajes mediante la serie de selector, además de **CORRELID** o **MSGID**. Hay que buscar en cada mensaje, por lo que si hay muchos mensajes en la cola, será caro.

Es más eficiente que la cola haga OPEN PUT PUT CLOSE en lugar de PUT1 PUT1.

Desencadenamiento en CICS

Cuando la frecuencia de llegada de mensajes para una cola desencadenada es baja, es eficiente utilizar primero el desencadenante. Cuando la velocidad de llegada de mensajes es de más de 10 mensajes por segundo, es más eficaz desencadenar la primera transacción y, a continuación, hacer que la transacción procese un mensaje y obtener el siguiente mensaje, etc. Si no llegan mensajes en un periodo corto (entre 0,1 y 1 segundo), la transacción finaliza. En un alto rendimiento, es posible que necesite varias transacciones en ejecución para procesar los mensajes e impedir que se acumulen. Cada mensaje desencadenante generado implica un put y un get de un mensaje desencadenante, lo cual duplica el coste del mensaje.

¿Cuántas conexiones o usuarios simultáneos se admiten?

Cada conexión utiliza almacenamiento virtual dentro del gestor de cola, por lo que cuantos más usuarios simultáneos haya, más almacenamiento se utiliza. Si necesita una agrupación de almacenamiento intermedio muy grande y un gran número de usuarios, es posible que esté restringido para el almacenamiento virtual, y quizás tenga que reducir el tamaño de las agrupaciones de almacenamiento intermedio.

Si se utiliza la seguridad, el gestor de colas almacena información en memoria caché dentro del gestor de colas durante un periodo largo. La cantidad de almacenamiento virtual que se utiliza dentro del gestor de colas se ve afectada.

CHINIT puede admitir hasta unas 10 000 conexiones. Esto está limitado por el almacenamiento virtual. Si una conexión utiliza más almacenamiento, por ejemplo, mediante TLS, el almacenamiento por conexión aumenta, lo que significa que **CHINIT** puede dar soporte a menos conexiones. Si está procesando mensajes grandes, estos requerirán más almacenamiento para almacenamientos intermedios en **CHINIT**, por lo que **CHINIT** puede dar soporte a menos mensajes.

Las conexiones con un gestor de colas remoto son más eficientes que las conexiones de cliente. Por ejemplo, cada solicitud de cliente MQ requiere dos flujos de red (uno para la solicitud y otro para la respuesta). Con un canal a un gestor de colas remoto, puede haber 50 envíos a través de la red antes de recibir una respuesta. Si está considerando una red de cliente grande, puede ser más eficiente utilizar un gestor de colas de concentrador en una caja distribuida y tener un canal de entrada y de salida del concentrador.

Otras cosas que afectan al rendimiento

El conjunto de datos de registro debe ser de al menos 1000 cilindros de tamaño. Si los registros son más pequeños, la actividad del punto de comprobación podría ser demasiado frecuente. En un sistema de alta ocupación, un punto de comprobación suele ser cada 15 minutos o más, con un rendimiento muy alto que puede algo menor. Cuando se produce un punto de comprobación, las agrupaciones de almacenamiento intermedio se exploran y los mensajes 'antiguos' y las páginas modificadas se escriben en el disco. Si los puntos de control son demasiado frecuentes, puede afectar al rendimiento. El valor de LOGLOAD también puede afectar a la frecuencia de punto de comprobación. Si el gestor de cola no finaliza con normalidad, al reiniciarlo podrían tener que leerse tres puntos de comprobación anteriores. El mejor intervalo de punto de comprobación es un equilibrio entre la actividad cuando se toma un punto de comprobación y la cantidad de datos de registro que es posible que haya que leer cuando se reinicia el gestor de colas.

Hay una sobrecarga significativa al iniciar un canal. Por lo general, es mejor iniciar un canal y dejarlo conectado, en lugar de frecuentes inicios y paradas.

Información relacionada

MP1K: IBM MQ for z/OS 9.0 Informe de rendimiento

z/OS IMS y las aplicaciones puente IMS en IBM MQ for z/OS

Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

- Para utilizar puntos de sincronización y llamadas MQI en aplicaciones IMS, consulte “Escritura de aplicaciones IMS utilizando IBM MQ” en la página 72.
- Para escribir aplicaciones que utilicen el puente IBM MQ - IMS, consulte “Escritura de aplicaciones puente IMS” en la página 76.

Consulte estos enlaces para obtener información adicional relativa a IMS y las aplicaciones puente IMS en IBM MQ for z/OS:

- “Escritura de aplicaciones IMS utilizando IBM MQ” en la página 72
- “Escritura de aplicaciones puente IMS” en la página 76

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Confirmación y restitución de unidades de trabajo” en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

z/OS Escritura de aplicaciones IMS utilizando IBM MQ

Hay consideraciones adicionales cuando se utiliza IBM MQ en aplicaciones IMS. Estas incluyen las llamadas de API de MQ que se pueden utilizar y el mecanismo utilizado para el punto de sincronización.

Utilice los siguientes enlaces para obtener más información sobre la escritura de aplicaciones IMS en IBM MQ for z/OS:

- “Puntos de sincronismo en aplicaciones IMS” en la página 73

- “Llamadas MQI en aplicaciones IMS” en la página 73

Restricciones

Hay restricciones sobre qué llamadas de API de IBM MQ puede utilizar una aplicación que utiliza el adaptador IMS.

Las siguientes llamadas de API de IBM MQ no están soportadas en una aplicación que utiliza el adaptador IMS:

- MQCB
- MQCB_FUNCTION
- MQCTL

Conceptos relacionados

“Escritura de aplicaciones puente IMS” en la página 76
Este tema contiene información sobre cómo escribir aplicaciones para utilizar el puente IBM MQ - IMS.

z/OS *Puntos de sincronismo en aplicaciones IMS*

En una aplicación IMS, puede establecer un punto de sincronismo utilizando las llamadas de IMS, tales como GU (obtener único) para IOPCB y CHKP (punto de comprobación).

Para restituir todos los cambios desde el punto de comprobación anterior, puede utilizar la llamada ROLB (retrotraer) de IMS. Para obtener más información, consulte llamada ROLB en la documentación de IMS .

El gestor de colas participa en un protocolo de confirmación de dos fases. El gestor de puntos de sincronismo de IMS es el coordinador.

El adaptador IMS cierra todos los descriptores abiertos en un punto de sincronismo, excepto en un entorno por lotes o en un entorno BMP no controlado por mensajes. Esto es debido a que un usuario diferente podría iniciar la siguiente unidad de trabajo y cuando se emiten las llamadas MQCONN, MQCONNX y MQOPEN, se realiza la comprobación de seguridad de IBM MQ, y no cuando se emiten las llamadas MQPUT o MQGET.

No obstante, en un entorno WFI (Wait-for-Input) o en un entorno PWFI (Pseudo Wait-for-Input), IMS no notifica a IBM MQ que cierre los descriptores hasta que llegue el mensaje siguiente ni se devuelve un código de estado QC a la aplicación. Si la aplicación está esperando en la región de IMS y cualquiera de estos manejadores pertenece a las colas desencadenadas, no se llevará a cabo el desencadenamiento porque las colas están abiertas. Por este motivo, las aplicaciones que se ejecutan en un entorno WFI o PWFI deben emitir MQCLOSE de forma explícita para los descriptores de colas antes de ejecutar GU para IOPCB para el mensaje siguiente.

Si una aplicación IMS, ya sea BMP o MPP, emite la llamada MQDISC, se cierran las colas abiertas pero no se toma implícitamente un punto de sincronismo. Si la aplicación finaliza con normalidad, se cierra cualquiera cola abierta y se lleva a cabo una confirmación implícita. Si la aplicación finaliza de forma anómala, se cierra cualquiera cola abierta y se lleva a cabo una restitución implícita.

z/OS *Llamadas MQI en aplicaciones IMS*

Utilice esta información para obtener información sobre el uso de las llamadas MQI en aplicaciones de servidor y aplicaciones de consulta.

Esta sección cubre el uso de llamadas MQI en los tipos siguientes de aplicaciones IMS:

- “Aplicaciones de servidor” en la página 73
- “Aplicaciones de consulta” en la página 76

Aplicaciones de servidor

A continuación se muestra el pseudocódigo de un modelo de aplicación de servidor MQI:

```
Initialize/Connect
```

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

El programa de ejemplo CSQ4ICB3 muestra la implementación, en C/370, de un BMP utilizando este modelo. El programa establece la comunicación con IMS primero y, después, con IBM MQ:

```

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

Return
```

La inicialización de IMS determina si se ha llamado el programa como un BMP orientado a mensajes u orientado a lotes y controla la conexión del gestor de colas IBM MQ y los descriptores de cola en consecuencia.

```

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

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

```
Return to calling function
```

La inicialización de IBM MQ se conecta al gestor de colas y abre las colas. En un BMP orientado a mensajes, este se invoca después de que se tome cada punto de sincronización de IMS: en un BMP orientado a lotes, este se invoca solo durante el inicio del programa:

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

Return to calling function
```

La implementación del modelo de servidor en un MPP (programa de procesamiento de mensajes) se ve influenciada por el hecho de que MPP procesa una única unidad de trabajo por invocación. Esto se debe a que, cuando se toma un punto de sincronización (GU), se cierran la conexión y los manejadores de cola, y se entrega el siguiente mensaje IMS. Esta limitación puede superarse en parte de una de las maneras siguientes:

- **Procesando muchos mensajes en una única unidad de trabajo.**

  Esto implica:

  – Leer un mensaje.
  – Procesar las actualizaciones necesarias.
  – Colocar la respuesta.

  en un bucle hasta que todos los mensajes se hayan procesado o hasta que se haya procesado el número máximo de mensajes, momento en el que se tomará un punto de sincronización.

  Este enfoque solo es válido para determinados tipos de aplicación (por ejemplo, una simple actualización o consulta de base de datos). Aunque los mensajes de respuesta MQI se pueden colocar con la autorización del originador del mensaje MQI que se está manejando, las implicaciones de seguridad de cualquier actualización de recurso IMS se deben abordar con cuidado.

- **Procesando un mensaje por invocación del MPP y garantizando una planificación múltiple del MPP para que procese todos los mensajes disponibles.**

  Utilice el programa de supervisor desencadenante de IBM MQ IMS (CSQQTRMN) para planificar la transacción MPP cuando haya mensajes en la cola de IBM MQ y no haya aplicaciones que le estén prestando servicio.

  Si el supervisor desencadenante inicia el MPP, el nombre del gestor de colas y el nombre de cola se pasan al programa, tal como se muestra en el siguiente fragmento de COBOL:

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
```

```
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME        ='
MQTMC-QNAME  OF MQTMC '='.
```

El modelo de servidor, que se espera que sea una tarea de larga ejecución, se soporta mejor en una región de procesamiento por lotes, aunque la BMP no se pueda desencadenar utilizando CSQQTRMN.

Aplicaciones de consulta

Una aplicación típica de IBM MQ que inicia una consulta o actualización funciona de la forma siguiente:

- Recopila datos del usuario.
- Coloca uno o más mensajes de IBM MQ.
- Obtiene los mensajes de respuesta (es posible que tenga que esperarlos)
- Proporciona una respuesta al usuario.

Puesto que los mensajes colocados en colas IBM MQ no pasan a estar disponibles en otras aplicaciones IBM MQ mientras no se confirman, se deben sacar del punto de sincronización, o la aplicación IMS se debe dividir en dos transacciones.

Si la consulta implica colocar un único mensaje, puede utilizar la opción *sin punto de sincronización*; sin embargo, si la consulta es más compleja, o si hay actualizaciones de recursos implicadas, puede que tenga problemas de coherencia si se produce un error y no utiliza puntos de sincronización.

Para superarlo, puede dividir las transacciones MPP de IMS utilizando llamadas MQI utilizando un conmutador de mensajes de programa a programa; consulte *IMS Intersystem Communication (ISC)* para obtener información sobre esto. Esto permite que un programa de consulta se implemente en un MPP:

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

z/OS Escritura de aplicaciones puente IMS

Este tema contiene información sobre cómo escribir aplicaciones para utilizar el puente IBM MQ - IMS.

Para obtener información sobre el puente IBM MQ - IMS, consulte El puente IMS.

Utilice los siguientes enlaces para obtener más información sobre la escritura de aplicaciones puente IMS en IBM MQ for z/OS:

- “Cómo el puente IMS maneja los mensajes” en la página 77
- “Escribir programas de transacciones IMS mediante IBM MQ” en la página 934

Conceptos relacionados

“Escritura de aplicaciones IMS utilizando IBM MQ” en la página 72
Hay consideraciones adicionales cuando se utiliza IBM MQ en aplicaciones IMS. Estas incluyen las llamadas de API de MQ que se pueden utilizar y el mecanismo utilizado para el punto de sincronización.

z/OS *Cómo el puente IMS maneja los mensajes*

Cuando utiliza el puente IBM MQ - IMS para enviar mensajes a una aplicación IMS, debe crear sus mensajes con un formato especial.

También debe colocar los mensajes en las colas IBM MQ que se han definido con una clase de almacenamiento que especifica el grupo XCF y el nombre de miembro del sistema IMS de destino. Esto se conoce como colas puente MQ-IMS o simplemente como colas **puente**.

El puente IBM MQ-IMS requiere acceso de entrada exclusivo (MQOO_INPUT_EXCLUSIVE) a la cola puente, si se ha definido con QSGDISP(QMGR), o si se ha definido con QSGDISP(SHARED) junto con la opción NOSHARE.

Un usuario no necesita iniciar sesión en IMS antes de enviar mensajes a una aplicación IMS. El ID de usuario del campo *UserIdentifier* de la estructura MQMD se utiliza para comprobar la seguridad. El nivel de comprobación se determina cuando IBM MQ se conecta a IMS y puede encontrar su descripción en la sección Control de acceso de aplicación para el puente IMS. Esto permite implementar un seudoinicio de sesión.

El puente IBM MQ - IMS acepta los siguientes tipos de mensajes:

- Mensajes que contienen datos de transacción de IMS y una estructura MQIIH. (Se describe en la sección MQIIH):

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

Nota:

1. Los corchetes, [ ], representan varios segmentos opcionales.
2. Establezca el campo *Format* de la estructura MQMD en MQFMT_IMS para utilizar la estructura MQIIH.

- Mensajes que contienen datos de transacciones IMS pero no la estructura MQIIH:

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

IBM MQ valida los datos del mensaje para asegurarse de que la suma de LL bytes más la longitud de MQIIH, si está presente, sea igual a la longitud del mensaje.

Cuando el puente IBM MQ - IMS obtiene mensajes de las colas puente, los procesa de este modo:

- Si el mensaje contiene una estructura MQIIH, el puente verifica el MQIIH, (consulte MQIIH), crea las cabeceras OTMA y envía el mensaje a IMS. El código de transacción se especifica en el mensaje de entrada. Si este es un LTERM, IMS responde con un mensaje DFS1288E. Si el código de transacción representa un mandato, IMS ejecuta el mandato. De lo contrario, el mensaje se pone en cola en IMS para la transacción.
- Si el mensaje contiene datos de la transacción IMS pero no así la estructura MQIIH, el puente IMS presupone lo siguiente:
  - El código de transacción está en los bytes 5 a 12 de los datos de usuario
  - La transacción está en modo no de conversación
  - La transacción está en el modo de confirmación 0 (confirmar y luego enviar)

– Se utiliza el valor de *Format* de MQMD como *MFSMapName* (en la entrada)
– El modo de seguridad es MQISS_CHECK

El mensaje de respuesta también se crea sin una estructura MQIIH, tomando el valor de *Format* para MQMD del *MFSMapName* de la salida de IMS.

El puente IBM MQ - IMS utiliza uno o dos Tpipes para cada cola IBM MQ:

- Se utiliza un Tpipe sincronizado para todos los mensajes que utilizan la modalidad de confirmación 0 (COMMIT_THEN_SEND) (estos se muestran con SYN en el campo de estado del mandato IMS /DIS TMEMBER client TPIPE xxxx)
- Se utiliza un Tpipe no sincronizado para todos los mensajes que utilizan el modo de confirmación 1 (SEND_THEN_COMMIT)

IBM MQ crea los Tpipes cuando se utilizan por primera vez. Un Tpipe existe hasta que se reinicia IMS. Los Tpipes sincronizados existen hasta que IMS se inicia en frío. No puede suprimir estos Tpipes manualmente.

Consulte los temas siguientes para obtener más información acerca de cómo maneja los mensajes el puente IBM MQ - IMS:

- "Correlación de mensajes de IBM MQ con los tipos de transacción IMS" en la página 78
- "Si el mensaje no se puede poner en la cola de IMS" en la página 79
- "Códigos feedback del puente IMS" en la página 79
- "Los campos MQMD de los mensajes del puente IMS" en la página 80
- "Los campos MQIIH de los mensajes del puente IMS" en la página 81
- "Mensajes de respuesta de IMS" en la página 82
- "Utilización de PCB de respuestas alternativas en las transacciones IMS" en la página 82
- "Envío de mensajes no solicitados desde IMS" en la página 82
- "Segmentación de mensajes" en la página 83
- "Conversión de datos para mensajes a y desde el puente IMS" en la página 83

Conceptos relacionados

"Escribir programas de transacciones IMS mediante IBM MQ" en la página 934
El código necesario para manejar las transacciones IMS mediante IBM MQ depende del formato de mensaje que requiera la transacción IMS y el rango de respuestas que pueda devolver. No obstante, hay varios puntos a tener en cuenta cuando su aplicación maneja información con formato de pantalla IMS.

z/OS *Correlación de mensajes de IBM MQ con los tipos de transacción IMS*

Una tabla que describe la correlación de mensajes de IBM MQ con los tipos de transacción IMS.

Tabla 4. Cómo se correlacionan los mensajes de IBM MQ con los tipos de transacción de IMS

Tipo de mensaje de IBM MQ	**Confirmar-luego-enviar (modalidad 0): utiliza Tpipes de IMS sincronizados**	**Enviar-luego-confirmar (modalidad 1): utiliza Tpipes de IMS no sincronizados**
Mensajes de IBM MQ persistentes	• Transacciones de funciones completas recuperables • Las transacciones no recuperables son rechazadas por IMS	• Transacciones de vía de acceso rápida • Transacciones conversacionales • Transacciones de funciones completas

Tabla 4. Cómo se correlacionan los mensajes de IBM MQ con los tipos de transacción de IMS (continuación)		
Tipo de mensaje de IBM MQ	**Confirmar-luego-enviar (modalidad 0): utiliza Tpipes de IMS sincronizados**	**Enviar-luego-confirmar (modalidad 1): utiliza Tpipes de IMS no sincronizados**
Mensajes de IBM MQ no persistentes	• Transacciones de funciones completas no recuperables • Las transacciones recuperables están permitidas con IMS V8 y APAR PQ61404 y todas las versiones posteriores de IMS	• Transacciones de vía de acceso rápida • Transacciones conversacionales • Transacciones de funciones completas

Nota: Los mandatos IMS no pueden utilizar mensajes IBM MQ persistentes con la modalidad de confirmación 0. Consulte Modalidad de confirmación (commitMode) para obtener más información.

z/OS *Si el mensaje no se puede poner en la cola de IMS*

Obtenga información sobre las acciones que se deben llevar a cabo si el mensaje no se puede poner en la cola de IMS.

Si el mensaje no se puede poner en la cola de IMS, IBM MQ toma la siguiente acción:

- Si un mensaje no se puede poner en IMS porque el mensaje no es válido, el mensaje se coloca en la cola de mensajes no entregados, y se envía un mensaje a la consola del sistema.
- Si el mensaje es válido, pero es rechazado por IMS, IBM MQ envía un mensaje de error a la consola del sistema, el mensaje incluye el código de detección de IMS y el mensaje de IBM MQ se coloca en la cola de mensajes no entregados. Si el código de detección de IMS es 001A, IMS envía un mensaje de IBM MQ que contiene el motivo de la anomalía en la cola de respuestas.

  Nota: En las circunstancias indicadas anteriormente, si IBM MQ no puede colocar el mensaje en la cola de mensajes no entregados por cualquier motivo, el mensaje se devuelve a la cola de IBM MQ de origen. Se envía un mensaje de error a la consola del sistema, y no se envían más mensajes desde esa cola.

  Para reenviar los mensajes, realice ***una*** de las acciones siguientes:

  – Detenga y reinicie Tpipes en el IMS correspondiente a la cola
  – Altere la cola a GET(DISABLED), y otra vez a GET(ENABLED)
  – Detenga y reinicie IMS o el OTMA
  – Detenga y reinicie el subsistema de IBM MQ
- Si IMS rechaza el mensaje para cualquier otra cosa que no sea un error de mensaje, el mensaje de IBM MQ se devuelve a la cola de origen, IBM MQ deja de procesar la cola y se envía un mensaje de error a la consola del sistema.

  Si hace falta un mensaje de informe de excepción, el puente lo coloca en la cola de respuesta con la autoridad del originador. Si no se puede poner el mensaje en la cola, el mensaje de informe se coloca en la cola de mensajes no entregados con la autorización del puente. Si no se puede poner en el DLQ, se descarta.

z/OS *Códigos feedback del puente IMS*

Normalmente, la salida de los códigos de detección IMS es en formato hexadecimal en los mensajes de la consola de IBM MQ, tal como CSQ2001I. Por ejemplo, el código de detección 0x001F. Los códigos feedback de IBM MQ como se muestran en la cabecera de los mensajes no entregados transferidos a la cola de mensajes no entregados son números decimales.

Los códigos de detección del puente IMS están en el rango de 301 a 399, o de 600 a 855 para el código de detección NACK 0x001A. Se correlacionan desde el código de detección IMS-OTMA de este modo:

1. El código de detección IMS-OTMA se convierte de un número hexadecimal a un número decimal.

2. 300 se añade al número resultante del cálculo de 1, lo que proporciona el código de IBM MQ *Feedback*.
3. El código de detección 0x001A de IMS-OTMA, decimal 26 es un caso especial. Se genera un código *Feedback* en el rango de 600-855.
   a. El código de razón IMS-OTMA se convierte de un número hexadecimal a un número decimal.
   b. 600 se añade al número resultante del cálculo de a, lo que proporciona el código de IBM MQ *Feedback*.

Para obtener información sobre los códigos de detección de IMS-OTMA, consulte Códigos de detección de OTMA para mensajes NAK.

z/OS *Los campos MQMD de los mensajes del puente IMS*

Obtenga información sobre los campos MQMD en los mensajes del puente IMS.

El MQMD del mensaje de origen es transportado por IMS en la sección Datos de usuario de las cabeceras de OTMA. Si el mensaje se origina en IMS, el MQMD se construye mediante la salida de resolución de destino de IMS. El MQMD de un mensaje recibido de IMS se crea de la forma siguiente:

StrucID
"MD "

Versión
MQMD_VERSION_1

Informe
MQRO_NONE

MsgType
MQMT_REPLY

Caducidad
Si MQIIH_PASS_EXPIRATION se establece en el campo Distintivos de MQIIH, este campo contiene el tiempo de caducidad restante; de lo contrario, se establece en MQEI_UNLIMITED

Comentarios
MQFB_NONE

Encoding
MQENC.Native (la codificación del sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (el ID del juego de caracteres codificados del sistema z/OS)

Formato
MQFMT_IMS si el MQMD.Format del mensaje de entrada es MQFMT_IMS, de lo contrario, IOPCB.MODNAME

Prioridad
MQMD.Priority del mensaje de entrada

Persistence
Depende de la modalidad de confirmación: MQMD.Persistence del mensaje de entrada si CM-1; la persistencia es igual a la recuperabilidad del mensaje de IMS si CM-0

MsgId
MQMD.MsgId si MQRO_PASS_MSG_ID, de lo contrario, nuevo MsgId (el valor predeterminado)

CorrelId
MQMD.CorrelId del mensaje de entrada si MQRO_PASS_CORREL_ID, de lo contrario, MQMD.MsgId del mensaje de entrada (valor predeterminado)

BackoutCount
0

ReplyToQ
Espacios en blanco

ReplyToQMgr
Espacios en blanco (establecido en el nombre qmgr local por el gestor de colas durante la MQPUT)

UserIdentifier
MQMD.UserIdentifier del mensaje de entrada

AccountingToken
MQMD.AccountingToken del mensaje de entrada

ApplIdentityData
MQMD.ApplIdentityData del mensaje de entrada

PutApplType
MQAT_XCF si no hay ningún error, de lo contrario MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> si no hay ningún error, de lo contrario, el nombre QMGR

PutDate
Fecha cuando se colocó el mensaje

PutTime
Hora cuando se colocó el mensaje

ApplOriginData
Espacios en blanco

z/OS *Los campos MQIIH de los mensajes del puente IMS*

Obtenga información sobre los campos MQIIH en los mensajes del puente IMS.

El MQIIH de un mensaje recibido de IMS se crea de la forma siguiente:

StrucId
"IIH "

Versión
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat del mensaje de entrada si MQIIH.ReplyToFormat no está en blanco; de lo contrario, IOPCB.MODNAME

Flags
0

LTermOverride
Nombre de LTERM (Tpipe) de la cabecera OTMA

MFSMapName
Nombre de correlación de la cabecera OTMA

ReplyToFormat
Espacios en blanco

Authenticator
MQIIH.Authenticator del mensaje de entrada si el mensaje de respuesta se está transfiriendo a una cola puente de MQ-IMS, de lo contrario, se deja en blanco.

TranInstanceId
ID de conversación / Señal de servidor de la cabecera OTMA, si está en la conversación. En las versiones de IMS anteriores a V14, este campo siempre adopta un valor nulo si no está en la

conversación. A partir de IMS V14 hacia adelante, este campo puede ser establecido por IMS aunque no esté en la conversación.

TranState
"C" si está en conversación, de lo contrario, en blanco

CommitMode
Modalidad de compromiso de la cabecera OTMA ("0" o "1")

SecurityScope
Espacio en blanco

Reserved
Espacio en blanco

z/OS *Mensajes de respuesta de IMS*

Cuando una transacción IMS realiza una llamada ISRT a su IOPCB, el mensaje se direcciona de nuevo al LTERM o TPIPE de origen.

Estos se ven en IBM MQ como mensajes de respuesta. Los mensajes de respuesta de IMS se colocan en la cola de respuestas especificada en el mensaje original. Si el mensaje no puede colocarse en la cola de respuesta, se coloca en la cola de mensajes no entregados utilizando la autorización del puente. Si el mensaje no puede colocarse en la cola de mensajes no entregados, se envía un acuse de recibo negativo a IMS indicando que no se puede recibir el mensaje. La responsabilidad del mensaje se devuelve entonces a IMS. Si utiliza la modalidad de confirmación 0, los mensajes desde ese Tpipe no se envían al puente y permanecen en la cola de IMS, es decir, no se envían más mensajes hasta que se reinicie. Si utiliza la modalidad de confirmación 1, otros trabajos pueden continuar.

Si la respuesta tiene una estructura MQIIH, su tipo de formato es MQFMT_IMS; de lo contrario, su tipo de formato se especifica mediante el nombre de MOD de IMS que se utiliza al insertar el mensaje.

z/OS *Utilización de PCB de respuestas alternativas en las transacciones IMS*

Cuando una transacción IMS utiliza PCB de respuestas alternativas (realiza una llamada ISRT al ALTPCB o emite una llamada CHNG a un PCB modificable), se invoca la salida previa al direccionamiento (DFSYPRX0) para determinar si el mensaje debe redireccionarse.

Si el mensaje se va a redireccionar, la salida de resolución de destino (DFSYDRU0) se invoca para confirmar el destino y preparar la información de cabecera. Consulte Utilización de salidas OTMA en IMS y La salida previa al direccionamiento DFSYPRX0 para obtener más información sobre estos programas de salida.

A menos que se realice una acción en las salidas, toda la salida de las transacciones IMS iniciadas desde un gestor de colas de IBM MQ, tanto a IOPCB como a ALTPCB, se devolverán al mismo gestor de colas.

z/OS *Envío de mensajes no solicitados desde IMS*

Para enviar mensajes desde IMS a una cola IBM MQ, tiene que invocar una transacción IMS que realice ISRT en un ALTPCB.

Tiene que escribir salidas de resolución de destino predireccionamiento para direccionar mensajes no solicitados desde IMS y crear los datos de usuario OTMA, para que el MQMD del mensaje se pueda crear correctamente. Consulte Salida de predireccionamiento DFSYPRX0 y Salida de usuario de resolución de destino para obtener información sobre estos programas de salida.

Nota: El puente IBM MQ - IMS no sabe si un mensaje que recibe es una respuesta o un mensaje no solicitado. Maneja el mensaje de la misma forma en cada caso, creando el MQMD y la MQIIH de la respuesta a partir del UserData de OTMA que llegan en el mensaje

Los mensajes no solicitados pueden crear nuevos Tpipes. Por ejemplo, si una transacción IMS existente ha conmutado a un nuevo LTERM (por ejemplo PRINT01), pero la implementación requiere que la salida se entregue a través de OTMA, se crea un nuevo Tpipe (llamado PRINT01 en este ejemplo). De forma predeterminada, se trata de un Tpipe no sincronizado. Si la implementación requiere que el mensaje sea recuperable, establezca el distintivo de salida de la salida de resolución de destino. Consulte la *Guía de personalización de IMS* para obtener más información.

z/OS *Segmentación de mensajes*

Puede definir las transacciones de IMS de modo que esperen una entrada de un único segmento o de varios segmentos.

La aplicación de IBM MQ de origen debe crear la entrada se salida utilizando la siguiente estructura MQIIH como uno o varios segmentos de datos LLZZ. Todos los segmentos de un mensaje de IMS deben estar contenidos en un único mensaje de IBM MQ que se envía con una única llamada MQPUT.

La longitud máxima de un segmento de datos LLZZ se define mediante IMS/OTMA (32767 bytes). La longitud total del mensaje de IBM MQ es la suma de todos los bytes LL más la longitud de la estructura MQIIH.

Todos los segmentos de la respuesta están contenidos en un único mensaje de IBM MQ.

Existe una restricción adicional en la limitación de 32 KB de los mensajes con el formato MQFMT_IMS_VAR_STRING. Cuando los datos de mensaje CCSID con ASCII combinado se convierten en un mensaje CCSID con EBCDIC combinado, se añade un byte de desplazamiento a teclado o un byte de desplazamiento desde teclado cada vez que hay una transacción entre los caracteres SBCS y DBCS. La restricción de 32 KB se aplica al tamaño máximo del mensaje. Esto es, dado que el campo LL del mensaje no puede superar los 32 KB, el mensaje no puede superar los 32 KB incluidos los caracteres de desplazamiento a o desde teclado. La aplicación que crea el mensaje debe tener esto en cuenta.

z/OS *Conversión de datos para mensajes a y desde el puente IMS*

La conversión de datos se lleva a cabo mediante la función de gestión de colas distribuidas, que puede llamar a cualquier salida necesaria, o mediante el agente de transferencia a colas dentro del grupo, que no da soporte al uso de salidas, cuando transfiere un mensaje a una cola de destino que tiene definida información XCF para su clase de almacenamiento. La conversión de datos no se produce cuando se entrega un mensaje a una cola mediante publicación/suscripción.

Cualquier salida debe estar disponible para el recurso de gestión de colas distribuidas del conjunto de datos al que hace referencia la sentencia CSQXLIB DD. Esto significa que puede enviar mensajes a una aplicación IMS utilizando el puente IBM MQ - IMS desde cualquier plataforma de IBM MQ.

Si hay errores de conversión, el mensaje se transfiere a la cola sin conversión. Finalmente, esto hace que el puente IBM MQ - IMS lo considere un error, ya que el puente no puede reconocer el formato de cabecera. Si se produce un error de conversión, el mensaje se envía a la consola de z/OS.

Consulte la sección “Escribir salidas de conversión de datos” en la página 1009 para obtener información detallada acerca de la conversión de datos en general.

Envío de mensajes al puente IBM MQ - IMS

Para asegurarse de que la conversión se realiza correctamente, debe indicar al gestor de colas el formato del mensaje.

Si el mensaje tiene una estructura MQIIH, el campo *Format* de MQMD debe establecerse en el formato incluido MQFMT_IMS y el campo *Format* de MQIIH debe establecerse en el nombre del formato que describe los datos de su mensaje. Si no existe ningún MQIIH, establezca el campo *Format* de MQMD en su nombre de formato.

Si sus datos, excepto LLZZ, son todos datos de caracteres (MQCHAR), utilice como nombre de formato, en MQIIH o MQMD, según convenga, el formato MQFMT_IMS_VAR_STRING incluido. De lo contrario, utilice su propio nombre de formato, en cuyo caso debe proporcionar también la salida de conversión de datos para su formato. La salida debe manejar la conversión de los LLZZ de su mensaje, además de los propios datos, pero no tiene que manejar ningún MQIIH en el inicio del mensaje.

Si su aplicación utiliza *MFSMapName*, en su lugar, puede utilizar mensajes con MQFMT_IMS y definir el nombre de correlación que se ha pasado a la transacción IMS en el campo MFSMapName de MQIIH.

Recepción de mensajes desde el puente IBM MQ - IMS

Si está presente una estructura MQIIH en el mensaje original que envía a IMS, también estará presente una estructura en el mensaje de respuesta.

Para asegurarse de que la respuesta se convierte correctamente:

- Si tiene una estructura MQIIH en su mensaje original, especifique el formato que desea para su mensaje de respuesta en el campo *ReplytoFormat* de MQIIH del mensaje original. Este valor se sustituye en el campo *Format* de MQIIH del mensaje de respuesta. Esto resulta especialmente útil si todos sus datos tienen el formato LLZZ<datos caracteres>.
- Si no tiene una estructura MQIIH en su mensaje original, especifique el formato que desea para el mensaje de respuesta, como el nombre MFS MOD del ISRT de la aplicación IMS en IOPCB.

Desarrollo de aplicaciones JMS/Jakarta Messaging y Java

IBM MQ proporciona tres interfaces de lenguaje Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSy IBM MQ classes for Java.

Acerca de esta tarea

V 9.3.0 JM 3.0 V 9.3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging es un proveedor de Jakarta Messaging que implementa las interfaces de Jakarta Messaging para IBM MQ como el sistema de mensajería. Jakarta Connectors Architecture proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno de Jakarta EE a un sistema de información empresarial (EIS) como, por ejemplo, IBM MQ o Db2.

Para obtener más información, consulte "¿Por qué debo utilizar IBM MQ classes for Jakarta Messaging?" en la página 86 y "Acceso a IBM MQ desde Java -Opción de API" en la página 89.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS es un proveedor de JMS que implementa las interfaces de JMS para IBM MQ como el sistema de mensajería. Java Platform, Enterprise Edition Connector Architecture (JCA) proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno Java EE con un EIS (Enterprise Information System) como IBM MQ o Db2.

Para obtener más información, consulte "¿Por qué debo utilizar IBM MQ classes for JMS?" en la página 87 y "Acceso a IBM MQ desde Java -Opción de API" en la página 89.

IBM MQ classes for Java

IBM MQ classes for Java permite utilizar IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

IBM MQ classes for Java encapsula la interfaz de cola de mensajes (MQI), la API nativa de IBM MQ , y utiliza el mismo modelo de objeto que otras interfaces orientadas a objetos, mientras que IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging implementan interfaces de mensajería de Java de Oracle y Java Community Process respectivamente.

Para obtener más información, consulte "¿Por qué debo utilizar IBM MQ classes for Java?" en la página 357 y "Acceso a IBM MQ desde Java -Opción de API" en la página 89.

Nota:

Stabilized IBM no hará mejoras adicionales en IBM MQ classes for Java y sus funciones se estabilizarán en el nivel suministrado en IBM MQ 8.0. Las aplicaciones existentes que utilizan IBM MQ classes for Java siguen recibiendo soporte completo, pero no se añadirán nuevas características y se rechazarán las solicitudes de mejoras. Soporte completo significa que los defectos se solucionarán junto con los cambios necesarios por los cambios en los requisitos del sistema IBM MQ.

IBM MQ classes for Java no se admite en IMS.

IBM MQ classes for Java no se admite en WebSphere Liberty. No se deben utilizar con la característica de mensajería de IBM MQ Liberty o con el soporte de JCA genérico. Para obtener más información, consulte Utilización de interfaces WebSphere MQ Java en entornos J2EE/JEE.

Utilización de IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de JMS 2.0 como la API de Jakarta Messaging 3.0 en la misma aplicación.

Nota: Para Jakarta Messaging 3.0, el control de la especificación JMS se mueve de Oracle a Java Community Process. Sin embargo, Oracle conserva el control del nombre "javax", que se utiliza en otras tecnologías Java que no se han movido al proceso de comunidad de Java . Por lo tanto, aunque Jakarta Messaging 3.0 es funcionalmente equivalente a JMS 2.0 , existen algunas diferencias en la denominación:

- El nombre oficial de la versión 3.0 es Jakarta Messaging en lugar de Java Message Service.
- Los nombres de paquete y constante tienen el prefijo `jakarta` en lugar de `javax`. Por ejemplo, en JMS 2.0 la conexión inicial con un proveedor de mensajería es un objeto `javax.jms.Connection` y en Jakarta Messaging 3.0 es un objeto `jakarta.jms.Connection` .

JMS 2.0 Los paquetes javax.jms definen las interfaces JMS y un proveedor JMS implementa estas interfaces para un producto de mensajería específico. IBM MQ classes for JMS es un proveedor de JMS que implementa interfaces JMS para IBM MQ.

JM 3.0 Los paquetes jakarta.jms definen las interfaces Jakarta Messaging y un proveedor Jakarta Messaging implementa estas interfaces para un producto de mensajería específico. IBM MQ classes for Jakarta Messaging es un proveedor de Jakarta Messaging que implementa interfaces Jakarta Messaging para IBM MQ.

Las especificaciones JMS y Jakarta Messaging esperan que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene objetos administrados en un repositorio central, y una aplicación JMS o Jakarta Messaging recupera estos objetos utilizando Java Naming Directory Interface (JNDI).

JMS 2.0 Para JMS 2.0, un administrador puede utilizar la herramienta de administración de IBM MQ JMS **JMSAdmin**, o IBM MQ Explorer, para crear y mantener objetos administrados en un repositorio central.

JM 3.0 Para Jakarta Messaging 3.0, no puede administrar JNDI utilizando IBM MQ Explorer. La administración JNDI está soportada por la variante Jakarta Messaging 3.0 de **JMSAdmin**, que es **JMS30Admin**.

Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

IBM MQ classes for JMS también proporciona dos conjuntos de extensiones a la API de JMS. El punto central de interés de estas extensiones es crear y configurar fábricas de conexiones y destinos de forma dinámica durante la ejecución, pero las extensiones también proporcionan una función que no está directamente relacionada con la mensajería como, por ejemplo, una función para la determinación de problemas.

Las extensiones IBM MQ JMS

IBM MQ classes for JMS contiene extensiones que se implementan en objetos como MQConnectionFactory, MQQueue y MQTopic. Estos objetos tienen propiedades y métodos que son específicos de IBM MQ. Los objetos pueden ser objetos administrados, o una aplicación puede

crearlos dinámicamente en tiempo de ejecución. Estas extensiones se denominan extensiones de IBM MQ JMS .

Las extensiones IBM JMS

IBM MQ classes for JMS también proporciona un conjunto más genérico de extensiones para la API de JMS , que no son específicas de IBM MQ como el sistema de mensajería o Java como el lenguaje de programación utilizado. Estas extensiones se denominan extensiones de IBM JMS y tienen los siguientes objetivos generales:

- Para proporcionar un mayor nivel de coherencia entre los proveedores de IBM JMS .
- Para facilitar la escritura de una aplicación puente entre dos sistemas de mensajería de IBM .
- Para facilitar el puerto de una aplicación de un proveedor de IBM JMS a otro.

Las extensiones proporcionan funciones que son similares a las proporcionadas en IBM MQ Message Service Client (XMS) for C/C++ y IBM MQ Message Service Client (XMS) for .NET.

Conceptos relacionados

Interfaces de lenguaje de IBM MQ Java

Tareas relacionadas

“Escritura de aplicaciones IBM MQ classes for JMS/Jakarta Messaging” en la página 145

Después de una breve introducción al modelo JMS , esta sección proporciona una guía detallada sobre cómo escribir aplicaciones IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging .

V 9.3.0 JM 3.0 V 9.3.0 ¿Por qué debo utilizar IBM MQ classes for Jakarta Messaging?

El uso de IBM MQ classes for Jakarta Messaging tiene varias ventajas, incluida la posibilidad de reutilizar los conocimientos de Jakarta Messaging existentes en su organización, y las aplicaciones son más independientes del proveedor de Jakarta Messaging y de la configuración subyacente de IBM MQ.

Resumen de las ventajas de utilizar IBM MQ classes for Jakarta Messaging

El uso de IBM MQ classes for Jakarta Messaging le permite reutilizar conocimientos existentes de Jakarta Messaging y proporcionar independencia de la aplicación.

- Puede reutilizar las habilidades y técnicas de Jakarta Messaging.

  IBM MQ classes for Jakarta Messaging es un proveedor de Jakarta Messaging que implementa las interfaces de Jakarta Messaging para IBM MQ como el sistema de mensajería. Si su organización es nueva en IBM MQ, pero ya tiene conocimientos de desarrollo de aplicaciones de Jakarta Messaging (o JMS), es posible que le resulte más fácil utilizar la API Jakarta Messaging familiar para acceder a los recursos de IBM MQ en lugar de utilizar una de las otras API proporcionadas con IBM MQ.
- Jakarta Messaging es una parte integral de Jakarta EE.

  Jakarta Messaging es la API natural que se debe utilizar para la mensajería en la plataforma Jakarta EE. Todo servidor de aplicaciones que sea compatible con Jakarta EE debe incluir un proveedor de Jakarta Messaging. Puede utilizar Jakarta Messaging en clientes de aplicaciones, servlets, Java Server Pages (JSP), enterprise Java beans (EJB) y beans controlados por mensajes (MDB). Tenga en cuenta, en particular, que las aplicaciones de Jakarta EE utilizan MDB para procesar mensajes de forma asíncrona, y todos los mensajes se entregan a los MDB como mensajes de Jakarta Messaging.
- Las fábricas de conexiones y los destinos se pueden almacenar como objetos administrados de Jakarta Messaging en un repositorio central en lugar de codificarse de forma fija en una aplicación.

  Un administrador puede crear y mantener objetos administrados de Jakarta Messaging en un repositorio central, y las aplicaciones IBM MQ classes for Jakarta Messaging pueden recuperar estos objetos utilizando Java Naming Directory Interface (JNDI). Las fábricas de conexiones y destinos de Jakarta Messaging encapsulan información específica de IBM MQcomo, por ejemplo, nombres de gestor de colas, nombres de canal, opciones de conexión, nombres de cola y nombres de tema. Si las fábricas de conexiones y destinos se almacenan como objetos administrados, esta información no se codifica de

forma fija en una aplicación. De este modo, se proporciona a la aplicación un grado de independencia respecto a la configuración subyacente de IBM MQ.

- Jakarta Messaging es una API estándar de la industria que puede proporcionar portabilidad de aplicaciones.

  Una aplicación Jakarta Messaging puede utilizar JNDI para recuperar fábricas de conexiones y destinos que se almacenan como objetos administrados, y utilizar sólo las interfaces definidas en el paquete `jakarta.jms` (Jakarta Messaging 3.0) para realizar operaciones de mensajería. Así, la aplicación es completamente independiente de cualquier proveedor Jakarta Messaging, como IBM MQ classes for Jakarta Messaging, y se puede portar de un proveedor Jakarta Messaging a otro sin tener que realizar ningún cambio en la aplicación.

  Si JNDI no está disponible en un entorno de aplicación determinado, una aplicación IBM MQ classes for Jakarta Messaging puede utilizar extensiones de la API de Jakarta Messaging para crear y configurar fábricas de conexiones y destinos dinámicamente en tiempo de ejecución. Así, la aplicación está completamente autocontenida, pero está vinculada a IBM MQ classes for Jakarta Messaging como el proveedor de Jakarta Messaging.

- Es posible que sea más fácil escribir aplicaciones puente mediante Jakarta Messaging.

  Una aplicación puente es una aplicación que recibe mensajes de un sistema de mensajería y los envía a otro sistema de mensajería. Escribir una aplicación de puente puede ser complicado si se usan las API y formatos de mensaje específicos del producto. En cambio, puede escribir una aplicación puente mediante dos proveedores Jakarta Messaging, uno para cada sistema de mensajería. Entonces la aplicación sólo utiliza una API, la API de Jakarta Messaging, y solo procesa los mensajes Jakarta Messaging.

Entornos desplegables

Para proporcionar la integración con un servidor de aplicaciones de Jakarta EE, los estándares de Jakarta EE requieren que los proveedores de mensajería suministren un adaptador de recursos. Siguiendo la especificación Jakarta Connectors Architecture , IBM MQ proporciona un adaptador de recursos que utiliza Jakarta Messaging para proporcionar funciones de mensajería dentro de cualquier entorno de Jakarta EE certificado. Para obtener más información, consulte “Liberty y el adaptador de recursos de IBM MQ” en la página 454.

Nota: WebSphere Application Server traditional no da soporte actualmente a Jakarta EE.

Fuera del entorno de Jakarta EE, se proporcionan los archivos de OSGi y JAR, lo que le facilita la obtención del IBM MQ classes for Jakarta Messaging. Estos archivos JAR son más fácilmente desplegables, ya sea autónomos o dentro de infraestructuras de gestión de software como, por ejemplo, Maven. Para obtener más información, consulte “Obtención de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging por separado” en la página 131.

Conceptos relacionados

IBM MQ para Jakarta Messaging: visión general

“Acceso a IBM MQ desde Java -Opción de API” en la página 89
IBM MQ proporciona tres interfaces de lenguaje Java .

JMS 2.0 ¿Por qué debo utilizar IBM MQ classes for JMS?

El uso de IBM MQ classes for JMS tiene varias ventajas, incluida la posibilidad de reutilizar los conocimientos de JMS existentes en su organización, y las aplicaciones son más independientes del proveedor de JMS y de la configuración subyacente de IBM MQ.

Resumen de las ventajas de utilizar IBM MQ classes for JMS

El uso de IBM MQ classes for JMS le permite reutilizar conocimientos existentes de JMS y proporcionar independencia de la aplicación.

V 9.3.0 V 9.3.0

Nota: JMS 2.0 ha sido reemplazado por Jakarta Messaging. IBM MQ classes for JMS sigue dando soporte al estándar JMS 2.0 , pero las futuras mejoras en la mensajería de Java solo surgirán en Jakarta Messaging, por lo tanto en IBM MQ classes for Jakarta Messaging. Los IBM MQ classes for JMS sólo se recomiendan para mantener y ampliar las aplicaciones JMS 2.0 existentes. IBM MQ classes for Jakarta Messaging debe ser la tecnología preferida para el nuevo desarrollo.

- Puede reutilizar las habilidades y técnicas de JMS.

  IBM MQ classes for JMS es un proveedor de JMS que implementa las interfaces de JMS para IBM MQ como el sistema de mensajería. Si su organización es nueva en IBM MQ, pero ya tiene conocimientos de desarrollo de aplicaciones de JMS, es posible que le resulte más fácil utilizar la API de JMS que ya conoce, para acceder a los recursos de IBM MQ, en lugar de una de las otras API que se proporcionan con IBM MQ.
- JMS es una parte integral de Java Platform, Enterprise Edition (Java EE).

  JMS es la API natural que se debe utilizar para la mensajería en la plataforma Java EE. Todo servidor de aplicaciones que sea compatible con Java EE debe incluir un proveedor de JMS. Puede utilizar JMS en clientes de aplicaciones, servlets, Java Server Pages (JSP), enterprise Java beans (EJB) y beans controlados por mensajes (MDB). Tenga en cuenta, en particular, que las aplicaciones de Java EE utilizan MDB para procesar mensajes de forma asíncrona, y todos los mensajes se entregan a los MDB como mensajes de JMS.
- Las fábricas de conexiones y los destinos se pueden almacenar como objetos administrados de JMS en un repositorio central en lugar de codificarse de forma fija en una aplicación.

  Un administrador puede crear y mantener objetos administrados de JMS en un repositorio central, y las aplicaciones IBM MQ classes for JMS pueden recuperar estos objetos utilizando Java Naming Directory Interface (JNDI). Las fábricas de conexiones y destinos de JMS encapsulan información específica de IBM MQcomo, por ejemplo, nombres de gestor de colas, nombres de canal, opciones de conexión, nombres de cola y nombres de tema. Si las fábricas de conexiones y destinos se almacenan como objetos administrados, esta información no se codifica de forma fija en una aplicación. De este modo, se proporciona a la aplicación un grado de independencia respecto a la configuración subyacente de IBM MQ.
- JMS es una API estándar de la industria que puede proporcionar portabilidad de aplicaciones.

  Una aplicación JMS puede utilizar JNDI para recuperar fábricas de conexiones y destinos que se almacenan como objetos administrados, y utilizar sólo las interfaces definidas en el paquete `javax.jms` para realizar operaciones de mensajería. Así, la aplicación es completamente independiente de cualquier proveedor JMS, como IBM MQ classes for JMS, y se puede portar de un proveedor JMS a otro sin tener que realizar ningún cambio en la aplicación.

  Si JNDI no está disponible en un entorno de aplicaciones determinado, una aplicación de IBM MQ classes for JMS puede utilizar extensiones en la API de JMS para crear y configurar dinámicamente fábricas de conexiones y destinos en tiempo de ejecución. Así, la aplicación está completamente autocontenida, pero está vinculada a IBM MQ classes for JMS como el proveedor de JMS.
- Es posible que sea más fácil escribir aplicaciones puente mediante JMS.

  Una aplicación puente es una aplicación que recibe mensajes de un sistema de mensajería y los envía a otro sistema de mensajería. Escribir una aplicación de puente puede ser complicado si se usan las API y formatos de mensaje específicos del producto. En cambio, puede escribir una aplicación puente mediante dos proveedores JMS, uno para cada sistema de mensajería. Entonces la aplicación sólo utiliza una API, la API de JMS, y solo procesa los mensajes JMS.

Entornos desplegables

Para proporcionar la integración con un servidor de aplicaciones de Java EE, los estándares de Java EE requieren que los proveedores de mensajería suministren un adaptador de recursos. Siguiendo la especificación Java EE Connector Architecture (JCA) , IBM MQ proporciona un adaptador de recursos que utiliza JMS para proporcionar funciones de mensajería dentro de cualquier entorno certificado para Java EE.

Aunque ha sido posible utilizar IBM MQ classes for Java dentro de Java EE, esta API no se ha diseñado u optimizado para este propósito. Para obtener más información sobre las consideraciones de IBM MQ classes for Java en Java EE, consulte “Ejecución de aplicaciones de IBM MQ classes for Java en Java EE” en la página 358.

Fuera del entorno de Java EE, se proporcionan los archivos de OSGi y JAR, lo que le facilita la obtención del IBM MQ classes for JMS. Estos archivos JAR son más fácilmente desplegables, ya sea autónomos o dentro de infraestructuras de gestión de software como, por ejemplo, Maven. Para obtener más información, consulte “Obtención de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging por separado” en la página 131.

Conceptos relacionados

V 9.3.0 V 9.3.0 IBM MQ para Jakarta Messaging: visión general

“¿Por qué debo utilizar IBM MQ classes for Jakarta Messaging?” en la página 86
El uso de IBM MQ classes for Jakarta Messaging tiene varias ventajas, incluida la posibilidad de reutilizar los conocimientos de Jakarta Messaging existentes en su organización, y las aplicaciones son más independientes del proveedor de Jakarta Messaging y de la configuración subyacente de IBM MQ.

“Acceso a IBM MQ desde Java -Opción de API” en la página 89
IBM MQ proporciona tres interfaces de lenguaje Java .

Acceso a IBM MQ desde Java -Opción de API

IBM MQ proporciona tres interfaces de lenguaje Java .

- V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging permite que las aplicaciones escritas utilizando las API de Jakarta Messaging 3.0 utilicen IBM MQ como proveedor de mensajería.

Jakarta Messaging es la dirección estratégica para la mensajería en aplicaciones Java .

Jakarta Messaging 3.0 es funcionalmente equivalente a JMS 2.0, por lo que para obtener más información, consulte “Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85.

IBM MQ classes for JMS

IBM MQ classes for JMS permite que las aplicaciones escritas utilizando las API de JMS 2.0 utilicen IBM MQ como proveedor de mensajería.

Como Jakarta Messaging reemplaza JMS, se recomienda utilizar IBM MQ classes for JMS en aplicaciones existentes o en entornos (por ejemplo, WebSphere Application Server) que no dan soporte a Jakarta Messaging.

No está soportado utilizar IBM MQ classes for Jakarta Messaging y IBM MQ classes for JMS en la misma aplicación.

Para obtener más información, consulte “Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85.

IBM MQ classes for Java

Stabilized La otra API que las aplicaciones Java pueden utilizar para acceder a los recursos de IBM MQ es IBM MQ classes for Java, que proporciona una API orientada a IBM MQpara que los programas utilicen IBM MQ como proveedor de mensajería. Sin embargo, IBM MQ classes for Java se estabiliza funcionalmente en el nivel suministrado en IBM MQ 8.0. Para obtener más información, consulte “¿Por

qué debo utilizar IBM MQ classes for Java?" en la página 357. Aunque las aplicaciones existentes que utilizan IBM MQ classes for Java siguen teniendo soporte completo, las aplicaciones nuevas deben utilizar IBM MQ classes for Jakarta Messaging.

Características comunes de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging proporcionan acceso a las características de mensajería punto a punto y de publicación/suscripción de IBM MQ. Además de enviar mensajes de JMS que proporcionan soporte para el modelo de mensajería estándar de JMS, las aplicaciones también pueden enviar y recibir mensajes sin cabeceras adicionales y, por lo tanto, pueden interoperar con otras aplicaciones de IBM MQ, por ejemplo, aplicaciones de C MQI. El control completo de las cargas útiles de mensajes MQMD y MQ está disponible.

Otras características de IBM MQ, como la transmisión de mensajes, la secuenciación asíncrona y los mensajes de informe, también están disponibles.

Utilizando las clases de ayudante PCF suministradas, los mensajes de administración de PCF de IBM MQ se pueden enviar y recibir a través de la API de JMS y se pueden utilizar para administrar gestores de colas.

Las características que se han añadido recientemente a IBM MQ, como el consumo asíncrono y la reconexión automática, no están disponibles en IBM MQ classes for Java, pero están disponibles en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

Solicitud de mejoras

Si necesita acceder a las características de IBM MQ que no están disponibles a través de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, puede plantear una idea.

A continuación, IBM puede indicar si la implementación es posible en la implementación IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , o si hay una práctica recomendada que se puede seguir.

Para las características de mensajería adicionales, como IBM es un colaborador del estándar abierto, estas características se pueden elevar como parte del proceso de JCP. Esto sólo se aplicaría a Jakarta Messaging.

Información relacionada

Bienvenido a IBM Ideas Portal

Proceso de revisión de especificación JMS Java

Uso de JMS para enviar mensajes PCF

V 9.3.0 JM 3.0 V 9.3.0 Requisitos previos para IBM MQ classes for Jakarta Messaging

Este tema describe lo que necesita conocer antes de utilizar IBM MQ classes for Jakarta Messaging. Para desarrollar y ejecutar aplicaciones de IBM MQ classes for Jakarta Messaging, necesita determinados componentes de software como requisitos previos.

Para obtener información sobre los requisitos previos de IBM MQ classes for Jakarta Messaging, consulte Requisitos del sistema para IBM MQ.

Para desarrollar aplicaciones de IBM MQ classes for Jakarta Messaging, necesita un kit de desarrollo de software (SDK) de Java SE. Para conocer detalles sobre los JDK soportados por cada sistema operativo, consulte Requisitos del sistema para IBM MQ.

Para ejecutar aplicaciones de IBM MQ classes for Jakarta Messaging, necesita los componentes de software siguientes:

- Un gestor de colas de IBM MQ.

- Un entorno de ejecución de Java runtime environment (JRE) para cada sistema en el que ejecute aplicaciones.
- IBM i Para IBM i, Qshell, que es la opción 30 del sistema operativo.
- z/OS Para z/OS, z/OS UNIX System Services (z/OS UNIX).

El proveedor de JSSE de IBM incluye un proveedor criptográfico certificado por FIPS, por lo que se puede configurar por programa para la conformidad con FIPS 140-2 y utilizarlo de inmediato. Por lo tanto, la conformidad con FIPS 140-2 se puede soportar directamente desde IBM MQ classes for Jakarta Messaging.

El proveedor de JSSE de Oracle puede tener un proveedor criptográfico certificado por FIPS que esté configurado en él, pero esto no está listo para su uso inmediato y no está disponible para la configuración programática. Por lo tanto, en este caso, IBM MQ classes for Jakarta Messaging no puede habilitar la conformidad con FIPS 140-2 directamente. Es posible que pueda habilitar manualmente esa conformidad, pero IBM no puede proporcionar actualmente información de guía sobre este tema.

Puede utilizar las direcciones de Internet Protocol versión 6 (IPv6) en las aplicaciones IBM MQ classes for Jakarta Messaging si las direcciones IPv6 están soportadas por la máquina virtual Java (JVM) y la implementación TCP/IP en el sistema operativo. La herramienta de administración de IBM MQ Jakarta Messaging , **JMS30Admin**, también acepta direcciones IPv6 . Para obtener más información sobre esta herramienta, consulte Configuración de objetos JMS y Jakarta Messaging utilizando las herramientas de administración.

La herramienta de administración de IBM MQ JMS y IBM MQ Explorer utilizan Java Naming Directory Interface (JNDI) para acceder a un servicio de directorio, que almacena objetos administrados. Las aplicaciones de IBM MQ classes for Jakarta Messaging también pueden utilizar JNDI para recuperar objetos administrados a partir de un servicio de directorio.

Nota: Para Jakarta Messaging 3.0, no puede administrar JNDI utilizando IBM MQ Explorer. La administración JNDI está soportada por la variante Jakarta Messaging 3.0 de **JMSAdmin**, que es **JMS30Admin**.

Un proveedor de servicios es un código que proporciona acceso a un servicio de directorio correlacionando llamadas de JNDI con el servicio de directorio. Un proveedor de servicios del sistema de archivos en los archivos `fscontext.jar` y `providerutil.jar` se proporciona con IBM MQ classes for Jakarta Messaging. El proveedor de servicios del sistema de archivos proporciona acceso a un servicio de directorio basado en el sistema de archivos local.

Si tiene la intención de utilizar un servicio de directorio basado en un servidor LDAP, debe instalar y configurar un servidor LDAP, o bien tener acceso a un servidor LDAP existente. En particular, debe configurar el servidor LDAP para almacenar objetos de Java. Si desea obtener información sobre cómo instalar y configurar el servidor LDAP, consulte la documentación que se suministra con el servidor.

JMS 2.0 Requisitos previos para IBM MQ classes for JMS

Este tema describe lo que necesita conocer antes de utilizar IBM MQ classes for JMS. Para desarrollar y ejecutar aplicaciones de IBM MQ classes for JMS, necesita determinados componentes de software como requisitos previos.

Para obtener información sobre los requisitos previos de IBM MQ classes for JMS, consulte Requisitos del sistema para IBM MQ.

Para desarrollar aplicaciones de IBM MQ classes for JMS, necesita un kit de desarrollo de software (SDK) de Java SE. Para conocer detalles sobre los JDK soportados por cada sistema operativo, consulte Requisitos del sistema para IBM MQ.

Para ejecutar aplicaciones de IBM MQ classes for JMS, necesita los componentes de software siguientes:

- Un gestor de colas de IBM MQ.
- Un entorno de ejecución de Java runtime environment (JRE) para cada sistema en el que ejecute aplicaciones.

- IBM i Para IBM i, Qshell, que es la opción 30 del sistema operativo.
- z/OS Para z/OS, z/OS UNIX System Services (z/OS UNIX).

El proveedor de JSSE de IBM incluye un proveedor criptográfico certificado por FIPS, por lo que se puede configurar por programa para la conformidad con FIPS 140-2 y utilizarlo de inmediato. Por lo tanto, la conformidad con FIPS 140-2 se puede habilitar directamente desde IBM MQ classes for Java y IBM MQ classes for JMS.

El proveedor de JSSE de Oracle puede tener un proveedor criptográfico certificado por FIPS que esté configurado en él, pero esto no está listo para su uso inmediato y no está disponible para la configuración programática. Por lo tanto, en este caso, IBM MQ classes for Java y IBM MQ classes for JMS no pueden habilitar la conformidad con FIPS 140-2 directamente. Es posible que pueda habilitar manualmente esa conformidad, pero IBM no puede proporcionar actualmente información de guía sobre este tema.

Puede utilizar las direcciones de Internet Protocol versión 6 (IPv6) en las aplicaciones IBM MQ classes for JMS si las direcciones IPv6 están soportadas por la máquina virtual Java (JVM) y la implementación TCP/IP en el sistema operativo. La herramienta de administración de IBM MQ JMS (consulte Configuración de objetos JMS utilizando la herramienta de administración) también acepta direcciones IPv6.

La herramienta de administración de IBM MQ JMS y IBM MQ Explorer utilizan Java Naming Directory Interface (JNDI) para acceder a un servicio de directorio, que almacena objetos administrados. Las aplicaciones de IBM MQ classes for JMS también pueden utilizar JNDI para recuperar objetos administrados a partir de un servicio de directorio. Un proveedor de servicios es un código que proporciona acceso a un servicio de directorio correlacionando llamadas de JNDI con el servicio de directorio. Un proveedor de servicios del sistema de archivos en los archivos `fscontext.jar` y `providerutil.jar` se proporciona con IBM MQ classes for JMS. El proveedor de servicios del sistema de archivos proporciona acceso a un servicio de directorio basado en el sistema de archivos local.

Si tiene la intención de utilizar un servicio de directorio basado en un servidor LDAP, debe instalar y configurar un servidor LDAP, o bien tener acceso a un servidor LDAP existente. En particular, debe configurar el servidor LDAP para almacenar objetos de Java. Si desea obtener información sobre cómo instalar y configurar el servidor LDAP, consulte la documentación que se suministra con el servidor.

Instalación y configuración de IBM MQ classes for JMS/Jakarta Messaging

Esta sección describe los directorios y archivos que se crean al instalar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, y le indica cómo configurar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging después de la instalación.

Conceptos relacionados

“Utilización del adaptador de recursos de IBM MQ” en la página 448
El adaptador de recursos permite que las aplicaciones que se ejecutan en un servidor de aplicaciones accedan a recursos de IBM MQ. El adaptador de recursos da soporte a la comunicación de entrada y de salida.

¿Qué se instala para IBM MQ classes for JMS?

Se crean varios archivos y directorios cuando se instala IBM MQ classes for JMS. En Windows, se realizan algunas configuraciones durante la instalación estableciendo automáticamente variables de entorno. En otras plataformas, y en algunos entornos Windows, debe establecer variables de entorno para poder ejecutar aplicaciones de IBM MQ classes for JMS.

Para la mayoría de los sistemas operativos, IBM MQ classes for JMS se instala como un componente opcional cuando instala IBM MQ.

Para obtener más información sobre cómo instalar IBM MQ, consulte:

Multi Instalación de IBM MQ

z/OS Instalación de IBM MQ for z/OS

Importante: Aparte de los archivos JAR reubicables descritos en “Archivos JAR reubicables de IBM MQ classes for JMS/Jakarta Messaging” en la página 94, no se da soporte a la copia de los archivos JAR de IBM MQ classes for JMS o bibliotecas nativas en otras máquinas, o en una ubicación diferente en una máquina en la que se ha instalado IBM MQ classes for JMS .

Directorios de instalación

En la Tabla 5 en la página 93, se muestra dónde se instalan los archivos de IBM MQ classes for JMS en cada plataforma.

Tabla 5. Directorios de instalación de IBM MQ classes for JMS

Plataforma	Directorio
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/java

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

El directorio de instalación incluye el contenido siguiente:

- Los archivos JAR de IBM MQ classes for JMS , incluidos los archivos JAR reubicables, que se encuentran en el directorio *MQ_INSTALLATION_PATH*\java\lib .
- Las bibliotecas nativas de IBM MQ, que se utilizan en las aplicaciones que emplean la interfaz nativa Java.

  Las bibliotecas nativas de 32 bits se instalan en el directorio *MQ_INSTALLATION_PATH*\java\lib y las bibliotecas nativas de 64 bits se pueden encontrar en el directorio *MQ_INSTALLATION_PATH*\java\lib64.

  Para obtener más información sobre las bibliotecas nativas de IBM MQ, consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 100.
- Scripts adicionales, que se describen en “Scripts proporcionados con IBM MQ classes for JMS/Jakarta Messaging” en la página 128. Estos scripts se encuentran en el directorio *MQ_INSTALLATION_PATH*\java\bin.
- Las especificaciones de la API de IBM MQ classes for JMS. La herramienta Javadoc se ha utilizado para generar las páginas HTML que contienen las especificaciones de la API.

  Las páginas HTML están en el directorio *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses:

  – ALW En AIX, Linux, and Windows, este subdirectorio contiene las páginas HTML individuales.

  – IBM i En IBM i, las páginas HTML están en un archivo denominado wmqjms_javadoc.jar.

  – z/OS En z/OS, las páginas HTML están en un archivo denominado wmqjms_javadoc.jar.
- Soporte de OGSi. Los paquetes OSGi se instalan en el directorio java\lib\OSGi y se describen en “Soporte de OSGi con IBM MQ classes for JMS” en la página 129.
- El adaptador de recursos de IBM MQ , que se puede desplegar en cualquier servidor de aplicaciones compatible con Java Platform, Enterprise Edition 7 (Java EE 7) o Jakarta EE .

  El adaptador de recursos IBM MQ está en el directorio *MQ_INSTALLATION_PATH*\java\lib\jca; si desea más información, consulte “Utilización del adaptador de recursos de IBM MQ” en la página 448
- Windows En Windows, los símbolos que se pueden utilizar para la depuración se instalan en el directorio *MQ_INSTALLATION_PATH*\java\lib\symbols.

El directorio de instalación también incluye algunos archivos que pertenecen a otros componentes de IBM MQ.

Aplicaciones de ejemplo

JMS 2.0 Se proporcionan algunas aplicaciones de ejemplo con IBM MQ classes for JMS. La Tabla 6 en la página 94 muestra dónde se instalan las aplicaciones de ejemplo en cada plataforma.

JM 3.0 Para IBM MQ classes for Jakarta Messaging, se están preparando nuevos ejemplos.

JMS 2.0

Tabla 6. Directorios de ejemplos para IBM MQ classes for JMS

Plataforma	**Directorio**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*`/samp/jms`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\jms`
IBM i IBM i	`/QIBM/ProdData/mqm/java/samples/jms`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/jms`

En esta tabla, *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que se ha instalado IBM MQ.

Después de la instalación, es posible que deba realizar algunas tareas de configuración para compilar y ejecutar aplicaciones.

"Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97 describe la vía de acceso de clases necesaria para ejecutar aplicaciones IBM MQ classes for JMS de ejemplo. En este tema, también se describen los archivos JAR adicionales a los que se debe hacer referencia en circunstancias especiales y las variables de entorno que deben establecerse para ejecutar los scripts que se proporcionan con IBM MQ classes for JMS.

Para controlar propiedades como, por ejemplo, el rastreo y registro de una aplicación, debe proporcionar un archivo de propiedades de configuración. El archivo de propiedades de configuración de IBM MQ classes for JMS se describe en "El archivo de configuración IBM MQ classes for JMS/Jakarta Messaging" en la página 102.

Conceptos relacionados

Problemas al desplegar el adaptador de recursos

Tareas relacionadas

"Utilización de las aplicaciones de ejemplo IBM MQ classes for JMS" en la página 124
Las aplicaciones de ejemplo IBM MQ classes for JMS proporcionan una descripción general de las características comunes de la API JMS. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Archivos JAR reubicables de IBM MQ classes for JMS/Jakarta Messaging
Los archivos JAR reubicables se pueden mover a sistemas que necesitan ejecutar IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Importante:

- Aparte de los archivos JAR reubicables descritos en Archivos JAR reubicables, no se da soporte a la copia de los archivos JAR IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging o bibliotecas nativas en otras máquinas, o en una ubicación diferente en una máquina en la que se han instalado IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

- No incluya los archivos JAR reubicables dentro de las aplicaciones desplegadas en servidores de aplicaciones Java EE , como WebSphere Application Server o WebSphere Liberty. En estos entornos, el adaptador de recursos de IBM MQ debe desplegarse y utilizarse en su lugar. Tenga en cuenta que WebSphere Application Server incluye el adaptador de recursos de IBM MQ , por lo que no es necesario desplegarlo manualmente en este entorno.
- Para evitar conflictos de cargador de clases, no se recomienda empaquetar los archivos JAR reubicables dentro de varias aplicaciones dentro del mismo tiempo de ejecución de Java . En este escenario, haga que los archivos JAR reubicables de IBM MQ estén disponibles en la vía de acceso de clases del tiempo de ejecución de Java .
- Si está empaquetando los archivos JAR reubicables dentro de las aplicaciones, asegúrese de incluir todos los archivos JAR de requisito previo tal como se describe en Archivos JAR reubicables. También debe asegurarse de que tiene los procedimientos adecuados para actualizar los archivos JAR empaquetados como parte del mantenimiento de la aplicación, para asegurarse de que IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging siguen siendo actuales y de que se vuelven a mediar los problemas conocidos.

Archivos JAR reubicables

Dentro de una empresa, los archivos siguientes se pueden mover a sistemas que necesitan ejecutar IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging:

- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` "4" en la página 95
- V 9.3.5 `bcpkix-jdk18on.jar` "3" en la página 95
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` "4" en la página 95
- V 9.3.5 `bcprov-jdk18on.jar` "3" en la página 95
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` "4" en la página 95
- V 9.3.5 `bcutil-jdk18on.jar` "3" en la página 95
- JMS 2.0 `com.ibm.mq.allclient.jar` "1" en la página 95
- V 9.3.0 JM 3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` "2" en la página 95
- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`
- `fscontext.jar`
- V 9.3.3 `jackson-annotations.jar`
- V 9.3.3 `jackson-core.jar`
- V 9.3.3 `jackson-databind.jar`
- `jakarta.jms-api.jar`
- `jms.jar`
- `org.json.jar`
- `providerutil.jar`

Notas:

1. JMS 2.0 y JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery desde IBM MQ 9.3.5
4. Long Term Support y Continuous Delivery antes de IBM MQ 9.3.5

JMS archivos JAR

V 9.3.0 V 9.3.0 `jms.jar` contiene las interfaces JMS 1.1 y JMS 2.0 , que se denominan `javax.jms.*`.

V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms-api.jar` contiene las interfaces Jakarta Messaging 3.0 , que se denominan `jakarta.jms.*`.

fscontext.jar y providerutil.jar

Los archivos `fscontext.jar` y `providerutil.jar` son necesarios si la aplicación realiza búsquedas JNDI utilizando un contexto de sistema de archivos.

El proveedor de seguridad de Bouncy Castle y los archivos JAR de soporte de CMS

Los archivos del proveedor de seguridad Bouncy Castle y los archivos JAR de soporte de CMS son necesarios. Para obtener más información, consulte Soporte para JRE que no son deIBM con AMS.

V 9.3.5 Para Continuous Delivery desde IBM MQ 9.3.5, son necesarios los siguientes archivos JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS Para Long Term Support y Continuous Delivery antes de IBM MQ 9.3.5, son necesarios los siguientes archivos JAR:

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

org.json.jar

El archivo `org.json.jar` es necesario si la aplicación IBM MQ classes for JMS utiliza una CCDT en formato JSON.

com.ibm.mq.allclient.jar y com.ibm.mq.jakarta.client.jar

Los archivos `com.ibm.mq.allclient.jar` y `com.ibm.mq.jakarta.client.jar` contienen las clases IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging, IBM MQ classes for Javay PCF y Headers Classes. Si mueve este archivo JAR a una nueva ubicación, asegúrese de que realiza los pasos necesarios para mantener esta nueva ubicación con los nuevos fixpacks de IBM MQ . Además, asegúrese de que el uso de los archivos se da a conocer al soporte de IBM si está obteniendo un arreglo temporal.

Para determinar la versión de los archivos `com.ibm.mq.allclient.jar` y `com.ibm.mq.jakarta.client.jar`, utilice el mandato siguiente:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

El ejemplo siguiente muestra algunos datos de salida de este mandato:

```
C:\Archivos de programa\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
```

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

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar, jackson-core.jar y jackson-databind.jar

V 9.3.3

Los tres archivos JAR Jackson son necesarios si la aplicación IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging crea conexiones TLS seguras con un gestor de colas.

Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging
Para poder compilar y ejecutar aplicaciones IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , el valor de la variable de entorno **CLASSPATH** debe incluir el archivo JAR (Java Archive) IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Dependiendo de sus necesidades, puede ser necesario añadir otros archivos JAR a CLASSPATH. Para ejecutar los scripts proporcionados con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, se deben establecer otras variables de entorno.

Antes de empezar

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

Importante: No se da soporte al establecimiento de la opción *-Xbootclasspath* de Java para incluir IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Acerca de esta tarea

Para compilar y ejecutar aplicaciones IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , utilice el valor **CLASSPATH** para la plataforma y la versión de mensajería de Java , tal como se muestra en las tablas siguientes. Como alternativa, puede especificar la vía de acceso de clases en el mandato **java** en lugar de utilizar la variable de entorno.

JMS 2.0 Para IBM MQ classes for JMS, el valor incluye el directorio de ejemplos, para que pueda compilar y ejecutar las aplicaciones de ejemplo de IBM MQ classes for JMS .

JM 3.0 Para IBM MQ classes for Jakarta Messaging, se están preparando nuevos ejemplos.

JM 3.0

*Tabla 7. Valores de **CLASSPATH** para Jakarta Messaging 3.0 para compilar y ejecutar aplicaciones IBM MQ classes for Jakarta Messaging*	
Plataforma	**CLASSPATH establecer**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

*Tabla 8. Valores de **CLASSPATH** para JMS 2.0 para compilar y ejecutar aplicaciones IBM MQ classes for JMS , incluidas las aplicaciones de ejemplo*	
Plataforma	**Valor de CLASSPATH**
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

En estas tablas, *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ .

El manifiesto del archivo JAR `com.ibm.mq.jakarta.client.jar` o `com.ibm.mq.allclient.jar` contiene referencias a la mayoría de los otros archivos JAR que necesitan las aplicaciones IBM MQ classes for JMS , por lo que no es necesario añadir estos archivos JAR a la vía de acceso de clases. Estos archivos JAR incluyen los archivos necesarios para las aplicaciones que utilizan la interfaz Java Naming Directory Interface (JNDI) para recuperar objetos administrados de un servicio de directorio y las aplicaciones que utilizan la API de transacción de Java (JTA).

Pero debe incluir archivos JAR adicionales en la vía de acceso de clases en los casos siguientes:

- Si utiliza clases de salida de canal que implementan las interfaces de salida de canal definidas en el paquete `com.ibm.mq`, en lugar de las definidas en el paquete `com.ibm.mq.exits`, debe añadir el archivo JAR de IBM MQ classes for Java, `com.ibm.mq.jar`, a la vía de acceso de clases.
- Si la aplicación utiliza JNDI para recuperar objetos administrados de un servicio de directorio, debe añadir los archivos JAR siguientes a la vía de acceso de clases:

  - `fscontext.jar`
  - `providerutil.jar`
- Si la aplicación utiliza la JTA, también debe añadir `jta.jar` a la vía de acceso de clases.

Nota: Estos archivos JAR adicionales sólo son necesarios para compilar las aplicaciones, no para ejecutarlas.

Los scripts proporcionados con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging utilizan las variables de entorno siguientes:

MQ_JAVA_DATA_PATH
Esta variable de entorno especifica el directorio para la salida de anotaciones y de rastreo.

MQ_JAVA_INSTALL_PATH
Esta variable de entorno especifica el directorio donde está instalado IBM MQ classes for JMS.

MQ_JAVA_LIB_PATH
Esta variable de entorno especifica el directorio donde se almacenan las bibliotecas de IBM MQ classes for JMS , tal como se muestra en las tablas anteriores.

Procedimiento

- Windows

  En Windows, después de instalar IBM MQ, ejecute el mandato **setmqenv**.

  Si no ejecuta este mandato en primer lugar, es posible que aparezca el siguiente mensaje de error al emitir un mandato **dspmqver** :

  ```
  AMQ8351: IBM MQ no se ha configurado
  o la característica IBM MQ JRE no se ha instalado.
  ```

  Nota: Este mensaje es de esperar si no ha instalado IBM MQ Java runtime environment (JRE) (consulte Comprobación de requisitos previos de características adicionales de Windows.

- Linux AIX

  En sistemas AIX and Linux , establezca las variables de entorno usted mismo:

  JMS 2.0 Para JMS 2.0, utilice uno de los scripts siguientes para establecer las variables de entorno:

  - Si está utilizando una JVM de 32 bits, utilice el script `setjmsenv`.
  - Si está utilizando una JVM de 64 bits en un sistema AIX o Linux , utilice el script `setjmsenv64`.

  JM 3.0 Para Jakarta Messaging 3.0, utilice uno de los scripts siguientes para establecer las variables de entorno:

  - Si está utilizando una JVM de 32 bits, utilice el script `setjms30env`.
  - Si está utilizando una JVM de 64 bits, utilice el script `setjms30env64`.

  Estos scripts residen en el directorio *MQ_INSTALLATION_PATH*`/java/bin`, donde *MQ_INSTALLATION_PATH* representa el directorio de nivel superior en el que se ha instalado IBM MQ.

  Puede utilizar estos scripts de varias maneras. Puede utilizar el script como base para establecer las variables de entorno necesarias, tal como se muestra en la tabla, o añadirlas a `.profile` utilizando un editor de texto. Si tiene una instalación atípica, edite el contenido del script según sea necesario. Como alternativa, puede ejecutar el script en cada sesión desde la que se deban ejecutar los scripts de inicio de JMS. Si elige esta opción, tendrá que ejecutar el script en cada ventana de shell que inicie, durante el proceso de verificación de JMS :

  - JMS 2.0 Para JMS 2.0, escriba `. ./setjmsenv` o `../setjmsenv64`.
  - JM 3.0 Para Jakarta Messaging 3.0, escriba `. ./setjms30env` o `. ./setjms30env64`.

IBM i En IBM i, debe establecer la variable de entorno **QIBM_MULTI_THREADED** en Y. Puede ejecutar aplicaciones de varias hebras del mismo modo que ejecuta aplicaciones de hebra única. Para obtener más información, consulte Configuración de IBM MQ con Java y JMS.

Tareas relacionadas

"Utilización de las aplicaciones de ejemplo IBM MQ classes for JMS" en la página 124
Las aplicaciones de ejemplo IBM MQ classes for JMS proporcionan una descripción general de las características comunes de la API JMS. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Referencia relacionada

"Scripts proporcionados con IBM MQ classes for JMS/Jakarta Messaging" en la página 128
Se proporcionan varios scripts para ayudar con las tareas comunes que se deben realizar al utilizar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

Configuración de las bibliotecas JNI (Java Native Interface)
Las aplicaciones de las IBM MQ classes for JMS que se conectan a un gestor de colas utilizando el transporte de enlaces o que se conectan a un gestor de colas utilizando el transporte de cliente y utilizan los programas de salida escritos en lenguajes que no son Java, se deben ejecutar en un entorno que permita el acceso a las bibliotecas JNI (Java Native Interface).

Antes de empezar

Consulte Configuración del proveedor de mensajería con información de bibliotecas nativas de IBM MQ para obtener más información sobre cómo utilizar el entorno WebSphere Application Server.

Acerca de esta tarea

Para configurar este entorno, debe configurar la vía de acceso de la biblioteca del entorno, de modo que JVM (Java Virtual Machine) pueda cargar la biblioteca mqjbnd antes de que inicie la aplicación de las IBM MQ classes for JMS.

IBM MQ proporciona dos bibliotecas JNI (Java Native Interface):

mqjbnd
Esta biblioteca las utilizan las aplicaciones que se conectan a un gestor de colas utilizando el transporte de enlaces. Proporciona la interfaz entre las IBM MQ classes for JMS y el gestor de colas. La biblioteca mqjbnd que se instala con IBM MQ 9.3 se puede utilizar para la conexión con cualquier gestor de colas de IBM MQ 9.3 o anterior.

mqjexitstub02
Las IBM MQ classes for JMS cargan la biblioteca mqjexitstub02 cuando una aplicación se conecta a un gestor de colas utilizando el transporte de cliente y utiliza un programa de salida de canal escrito en un lenguaje que no es Java.

En determinadas plataformas, IBM MQ instala las versiones de 32 bits y 64 bits de estas bibliotecas JNI. En la Tabla 1, se muestra la ubicación de las bibliotecas para cada plataforma.

Tabla 9. La ubicación de las bibliotecas de IBM MQ classes for JMS para cada plataforma

Plataforma	**Directorio que contiene las bibliotecas de IBM MQ classes for JMS**
AIX AIX Linux Linux (plataformas POWER, x86-64 y zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)

Tabla 9. La ubicación de las bibliotecas de IBM MQ classes for JMS para cada plataforma (continuación)	
Plataforma	**Directorio que contiene las bibliotecas de IBM MQ classes for JMS**
Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliotecas de 64 bits)
z/OS	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 31 bits y 64 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Nota: z/OS En z/OS, puede utilizar una JVM (Java Virtual Machine) de 31 bits o 64 bits. No tiene que especificar qué bibliotecas JNI se han de utilizar. Las IBM MQ classes for JMS pueden determinar por su cuenta las bibliotecas JNI que se han de cargar.

Procedimiento

1. Configure la propiedad **`java.library.path`** de la JVM de uno de estos dos modos:
   - Especifique el argumento JVM como se muestra en el ejemplo siguiente:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux Por ejemplo, para una JVM de 64 bits en Linux en una instalación de ubicación predeterminada, especifique:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - Si configura el entorno de shell de modo que la JVM configure su propia `java.library.path`. Esta vía de acceso varía según la plataforma y la ubicación en la que ha instalado IBM MQ. Por ejemplo, para una JVM de 64 bits y una ubicación de instalación de IBM MQ predeterminada, puede utilizar los valores siguientes:

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

     Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   La siguiente es una pila de excepción que verá cuando el entorno no se ha configurado correctamente:

```
Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: No se ha podido cargar la biblioteca JNI de WebSphere MQ nativa: '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.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a: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(WMQConnectionF
actory.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. Una vez configurado el entorno de 32 bits o 64 bits, inicie la aplicación de las IBM MQ classes for JMS utilizando el mandato:

   ```
   java application-name
   ```

   donde *nombre_aplicación* es el nombre de la aplicación de las IBM MQ classes for JMS que se ha de ejecutar.

   IBM MQ classes for JMS emite una excepción que contiene el código de razón 2495 de IBM MQ (MQRC_MODULE_NOT_FOUND) si:

   - La aplicación de las IBM MQ classes for JMS se ejecuta en un Java runtime environment de 32 bits y se ha configurado un entorno de 64 bits para las IBM MQ classes for JMS, ya que el Java runtime environment de 32 bits no puede cargar la biblioteca nativa Java de 64 bits.
   - La aplicación de las IBM MQ classes for JMS se ejecuta en un Java runtime environment de 64 bits y se ha configurado un entorno de 32 bits para las IBM MQ classes for JMS, ya que el Java runtime environment de 64 bits no puede cargar la biblioteca nativa de Java de 32 bits.

El archivo de configuración IBM MQ classes for JMS/Jakarta Messaging
Los archivos de configuración IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging especifican propiedades que se utilizan para configurar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

Nota: Las propiedades definidas en el archivo de configuración también se pueden establecer como propiedades del sistema JVM. Si una propiedad se establece en el archivo de configuración y también se define como una propiedad del sistema, la propiedad del sistema tiene prioridad. Por lo tanto, si es necesario, puede alterar temporalmente cualquier propiedad contenida en el archivo de configuración especificando la propiedad como propiedad del sistema en el mandato **java**.

El formato de un archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging es el de un archivo de propiedades Java estándar. Se proporciona un archivo de configuración de ejemplo denominado `jms.config` en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS. Este archivo documenta todas las propiedades soportadas y sus valores predeterminados.

Puede elegir el nombre y la ubicación de un archivo de configuración de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Al iniciar la aplicación, utilice un mandato **java** con el formato siguiente:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

En el mandato, *config_file_url* es un localizador universal de recursos (URL) que especifica el nombre y la ubicación del archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Están soportados los URL de los tipos siguientes: http, file, ftp y jar.

Esto es un ejemplo de un mandato **java**:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Este mandato identifica el archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging como el archivo `D:\mydir\mjms.config` en el sistema Windows local.

Cuando se inicia una aplicación, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging lee el contenido del archivo de configuración y almacena las propiedades especificadas en un almacén de propiedades interno. Si el mandato **java** no identifica un archivo de configuración, o si no se puede encontrar el archivo de configuración, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utiliza los valores predeterminados para todas las propiedades.

Se puede utilizar un archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging con cualquiera de los transportes soportados entre una aplicación y un gestor de colas o intermediario.

Alteración temporal de las propiedades especificadas en un archivo de configuración de IBM MQ MQI client

Un archivo de configuración IBM MQ MQI client también puede especificar propiedades que se utilizan para configurar IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging. Pero las propiedades especificadas en un archivo de configuración de IBM MQ MQI client se aplican sólo cuando una aplicación se conecta a un gestor de colas en la modalidad de cliente.

Si es necesario, puede alterar temporalmente cualquier atributo en un archivo de configuración IBM MQ MQI client especificándolo como una propiedad en un archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Para alterar temporalmente un atributo en un archivo de configuración IBM MQ MQI client , utilice una entrada con el formato siguiente en el archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Las variables de la entrada tienen los significados siguientes:

stanza
Nombre de la stanza contenida en el archivo de configuración de IBM MQ MQI client donde reside el atributo.

propName
Nombre del atributo tal como está especificado en el archivo de configuración de IBM MQ MQI client.

propValue
Valor de la propiedad que altera temporalmente el valor del atributo especificado en el archivo de configuración de IBM MQ MQI client.

Como alternativa, puede alterar temporalmente un atributo contenido en un archivo de configuración de IBM MQ MQI client especificando el atributo como propiedad del sistema en el mandato **java**. Utilice el formato anterior para especificar el atributo como propiedad del sistema.

Sólo los atributos siguientes de un archivo de configuración IBM MQ MQI client son relevantes para IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging. Si especifica o altera temporalmente otros atributos, dicha acción no tendrá afecto. En particular, observe que no se utilizan los atributos `ChannelDefinitionFile` y `ChannelDefinitionDirectory` contenidos en la Stanza CHANNELS del archivo de configuración de cliente. Consulte “Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS” en la página 292 para obtener detalles sobre cómo utilizar la CCDT con IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Tabla 10. Stanzas del archivo de configuración de cliente y los atributos que contienen

Stanza	**Atributo**
Stanza CHANNELS del archivo de configuración de cliente	Put1DefaultAlwaysSync
Stanza CHANNELS del archivo de configuración de cliente	DefRecon
Stanza CHANNELS del archivo de configuración de cliente	ReconDelay

Tabla 10. Stanzas del archivo de configuración de cliente y los atributos que contienen (continuación)	
Stanza	**Atributo**
Stanza CHANNELS del archivo de configuración de cliente	PasswordProtection
Stanza ClientExitPath del archivo de configuración de cliente	Vía de acceso predeterminada de las salidas
Stanza ClientExitPath del archivo de configuración de cliente	ExitsDefaultPath64
Stanza ClientExitPath del archivo de configuración de cliente	JavaExitsClasspath
Stanza JMQI del archivo de configuración de cliente	useMQCSPauthentication
Stanza MessageBuffer del archivo de configuración de cliente	MaximumSize
Stanza MessageBuffer del archivo de configuración de cliente	PurgeTime
Stanza MessageBuffer del archivo de configuración de cliente	UpdatePercentage
Stanza TCP del archivo de configuración de cliente	ClntRcvBufSize
Stanza TCP del archivo de configuración de cliente	ClntSndBufSize
Stanza TCP del archivo de configuración de cliente	Connect_Timeout
Stanza TCP del archivo de configuración de cliente	KeepAlive

Para obtener más detalles sobre la configuración de IBM MQ MQI client , consulte el archivo de configuración de IBM MQ MQI client , `mqclient.ini`

Utilización del rastreo de Java Standard Environment para configurar el rastreo de JMS
Utilice la stanza Java Standard Environment Trace Settings para configurar el recurso de rastreo IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging .

`com.ibm.msg.client.commonservices.trace.outputName` = ***traceOutputName***
`traceOutputName` es el directorio y el nombre de archivo al que se envía la salida de rastreo.

De forma predeterminada, la información de rastreo se escribe en un archivo de rastreo en el directorio de trabajo actual de la aplicación. El nombre del archivo de rastreo depende del entorno en el que se está ejecutando la aplicación:

- V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, si la aplicación ha cargado IBM MQ classes for Jakarta Messaging desde el archivo JAR reubicable `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) o IBM MQ classes for JMS desde el archivo JAR reubicable `com.ibm.mq.allclient.jar` (JMS 2.0), el rastreo se graba en un archivo denominado `mqjavaclient_%PID%.cl%u.trc`.
- Desde IBM MQ 9.1.5 y IBM MQ 9.1.0 Fix Pack 5:
  - Si la aplicación ha cargado IBM MQ classes for JMS desde el archivo JAR reubicable `com.ibm.mq.allclient.jar`, el rastreo se graba en un archivo denominado `mqjavaclient_%PID%.cl%u.trc`.
  - Si la aplicación ha cargado IBM MQ classes for JMS desde el archivo JAR `com.ibm.mqjms.jar`, el rastreo se graba en un archivo denominado `mqjava_%PID%.cl%u.trc`.
- Desde IBM MQ 9.0.0 Fix Pack 2:

  - Si la aplicación ha cargado IBM MQ classes for JMS desde el archivo JAR reubicable `com.ibm.mq.allclient.jar`, el rastreo se graba en un archivo denominado `mqjavaclient_%PID%.trc`.
  - Si la aplicación ha cargado IBM MQ classes for JMS desde el archivo JAR `com.ibm.mqjms.jar`, el rastreo se graba en un archivo denominado `mqjava_%PID%.trc`.
- Para IBM MQ classes for JMS para IBM MQ 9.0.0 Fix Pack 1 o anterior, el rastreo se escribe en un archivo llamado `mqjms_%PID%.trc`.

donde *%PID%* es el identificador de proceso de la aplicación que se está rastreando, y *%u* es un número único para diferenciar los archivos entre las hebras que ejecutan los classloaders de Java.

Si un ID de proceso no está disponible, se genera un número aleatorio con la letra `f`como prefijo. Para incluir el ID de proceso en un nombre de archivo que especifique, utilice la serie `%PID%`.

Si especifica otro directorio, éste debe existir, y debe tener permisos de escritura para el mismo. Si no tiene permisos de escritura, la salida de rastreo se escribe en `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList es una lista de los paquetes y clases que se rastrean, o los valores especiales `ALL` o `NONE`.

Separe los nombres de paquete y clase con un punto y coma, `;`. *includeList* toma como valor predeterminado `ALL`y rastrea todos los paquetes y clases en IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Nota: Puede incluir un paquete pero, a continuación, excluya los subpaquetes de dicho paquete. Por ejemplo, si incluye el paquete `a.b` y excluye `a.b.x`, el rastreo incluye todo el contenido de `a.b.y` y `a.b.z`, pero no el contenido de `a.b.x` o `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList es una lista de paquetes y clases que no se rastrean, o los valores especiales `ALL` o `NONE`.

Separe los nombres de paquete y clase con un punto y coma, `;`. *excludeList* toma como valor predeterminado `NONE`y, por lo tanto, no excluye ningún paquete ni clase en IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging del rastreo.

Nota: Puede excluir un paquete pero, a continuación, incluir subpaquetes de dicho paquete. Por ejemplo, si excluye el paquete `a.b` e incluye el paquete `a.b.x`, el rastreo incluye todo el contenido de `a.b.x` y de `a.b.x.1` pero no así lo de `a.b.y` o `a.b.z`.

Cualquier paquete o clase que se haya especificado, en el mismo nivel, ya que se incluye los incluidos y los excluidos.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes es el número máximo de bytes que se rastrean de cualquier matriz de bytes.

Si *maxArrayBytes* se establece en un entero positivo, limita el número de bytes de la matriz de bytes que se escriben en el archivo de rastreo. Trunca la matriz de bytes tras escribir *maxArrayBytes*. Definir *maxArrayBytes* reduce el tamaño del archivo de rastreo resultante y reduce el efecto de rastrear el rendimiento de la aplicación.

Un valor `0` para esta propiedad indica que no se envía el contenido de ninguna de las matrices de bytes al archivo de rastreo.

El valor predeterminado es `-1`, que elimina cualquier límite en el número de bytes en una matriz de bytes que se envían al archivo de rastreo.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes es el número máximo de bytes que se escriben en un archivo de salida de rastreo.

maxTraceBytes funciona con *traceCycles*. Si el número de bytes de rastreo escritos se acerca al límite, el archivo se cierra y se inicia un nuevo archivo de salida de rastreo.

Un valor `0` significa que un archivo de salida de rastreo tiene longitud cero. El valor predeterminado es `-1`, lo que significa que la cantidad de datos que se grabarán en el archivo de salida de rastreo es ilimitada.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles es el número de archivos de rastreo de salida que hay que recorrer.

Si el archivo de salida de rastreo actual alcanza el límite especificado por *maxTraceBytes*, el archivo se cierra. La salida de rastreo adicional se graba en el archivo de salida de rastreo siguiente de la secuencia. Cada archivo de salida de rastreo se distingue por un sufijo numérico que se añade al nombre de archivo. El archivo de salida de rastreo actual o más reciente es `mqjms.trc.0`, el siguiente archivo de salida de rastreo más reciente es `mqjms.trc.1`. Los archivos de rastreo más antiguos siguen el mismo patrón de numeración hasta el límite.

El valor predeterminado de *traceCycles* es 1. Si *traceCycles* es 1, cuando el archivo de salida de rastreo actual alcanza su tamaño máximo, el archivo se cierra y se suprime. Se inicia un nuevo archivo de salida de rastreo con el mismo nombre. Por consiguiente, únicamente existe un archivo de salida de rastreo simultáneamente.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controla si los parámetros de método y valores de retorno se incluyen en el rastreo.

traceParameters es, de forma predeterminada, TRUE. Si *traceParameters* se establece en FALSE, sólo se rastrean las signaturas de método.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Existe una fase de inicialización de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging durante la cual se asignan los recursos. El recurso de rastreo principal se inicializa durante la fase de asignación de recursos.

Si *startup* se establece en TRUE, se utiliza el rastreo de inicio. La información de rastreo se produce inmediatamente e incluye la configuración de todos los componentes, incluido el propio recurso de rastreo. La información de rastreo inicial puede utilizarse para diagnosticar problemas de configuración. La información de rastreo de inicio siempre se graba en `System.err`.

startup es, de forma predeterminada, FALSE.

startup se comprueba antes de que finalice la inicialización. Por este motivo, especifique solo la propiedad en la línea de mandatos como una propiedad del sistema Java. No lo especifique en el archivo de configuración IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Establezca *compressedTrace* en TRUE para comprimir la salida de rastreo.

El valor predeterminado de *compressedTrace* es FALSE.

Si *compressedTrace* se establece en TRUE, la salida de rastreo se comprime. El nombre del archivo de salida de rastreo tiene la extensión `.trz`. Si la compresión se establece en FALSE, que es el valor predeterminado, el archivo tiene la extensión `.trc` para indicar que no está comprimido. Sin embargo, si el nombre de archivo para la salida de rastreo se ha especificado en *traceOutputName*, se utiliza dicho nombre; no se aplica ningún sufijo al archivo.

La salida de rastreo comprimida es más pequeña que la no comprimida. Dado que significa menos E/S, puede escribirse con mayor rapidez que el archivo no comprimido. El rastreo comprimido tiene menos efecto en el rendimiento de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging que el rastreo no comprimido.

Si se ha establecido *maxTraceBytes* y *traceCycles*, se crean varios archivos de rastreo comprimidos en lugar de varios archivos planos.

Si IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging finaliza de forma no controlada, es posible que un archivo de rastreo comprimido no sea válido. Por este motivo, la compresión de rastreo sólo se debe utilizar cuando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging se cierra de forma controlada. Utilice la compresión de rastreo solo si los problemas que se están

investigando no dan lugar a que la JVM se cierre de forma inesperada. No utilice la compresión de rastreo si diagnostica problemas que pueden dar lugar al cierre de `System.Halt()` o a una terminación no controlada de la JVM.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel especifica un nivel de filtrado distinto para el rastreo. Los niveles de rastreo definidos son los siguientes:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Cada nivel de rastreo incluye todos los niveles inferiores. Por ejemplo, si el nivel de rastreo se establece en `TRACE_INFO`, los puntos de rastreo con un nivel definido de `TRACE_EXCEPTION`, `TRACE_WARNING` o `TRACE_INFO` se escriben en el rastreo. Todos los demás puntos de rastreo se excluyen.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controla si se utiliza el servicio de rastreo de cliente de IBM MQ JMS en un entorno WebSphere Application Server.

Si *standaloneTrace* está establecido en TRUE, se utilizan las propiedades de rastreo del cliente de IBM MQ JMS para determinar la configuración del rastreo.

Si *standaloneTrace* está establecido en FALSE, y el cliente de IBM MQ JMS se está ejecutando en un contenedor de WebSphere Application Server, se utiliza el servicio de rastreo de WebSphere Application Server. La información de rastreo que se genera depende de los valores de rastreo del servidor de aplicaciones.

El valor predeterminado de *standaloneTrace* es FALSE.

Stanza Logging
Utilice la stanza Logging para configurar el recurso de registro de IBM MQ classes for JMS.

Las siguientes propiedades se pueden incluir en la stanza Logging:

com.ibm.msg.client.commonservices.log.outputName = path
El nombre del archivo de registro que utiliza el recurso de registro de IBM MQ classes for JMS. El valor predeterminado es `mqjms.log`, que se escribe en el directorio de trabajo actual para el entorno de ejecución Java donde se ejecuta IBM MQ classes for JMS.

La propiedad puede tomar uno de los valores siguientes:

- un nombre de vía de acceso único
- una lista separada por comas de nombres de vía de acceso (todos los datos se registran en todos los archivos)

Cada nombre de vía de acceso puede ser un nombre de vía de acceso absoluta o relativo o:

"stderr" o "System.err"
Representa la secuencia de errores estándar.

"stdout" o "System.out"
Representa la secuencia de salida estándar.

com.ibm.msg.client.commonservices.log.maxBytes
El número máximo de bytes que se registran desde cualquier llamada a los datos de mensajes de registro.

Entero positivo
Los datos se escriben hasta ese valor de bytes para cada llamada de registro.

0
No se escriben datos.

-1
Se escriben datos ilimitados (valor predeterminado).

com.ibm.msg.client.commonservices.log.limit
El número máximo de bytes escritos en cualquier archivo de 1 registro (el valor predeterminado es 262144).

Entero positivo
Los datos se escriben hasta ese valor de bytes para cada archivo de registro.

0
No se escriben datos.

-1
Se escriben datos ilimitados.

com.ibm.msg.client.commonservices.log.count
El número de archivos de registro por los que se realiza un ciclo. A medida que cada archivo alcanza `com.ibm.msg.client.commonservices.trace.limit`, se inicia el rastreo en el siguiente archivo, siendo el valor predeterminado 3.

Entero positivo
El número de archivos por los que se realiza un ciclo.

0
Un archivo individual.

La stanza Java SE Specifics
Utilice la stanza Java SE Specifics para configurar las propiedades que se utilizan cuando se utiliza IBM MQ classes for JMS en un entorno de Java Standard Edition.

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE|FALSE
Determina si un archivo de JavaCore se escribe inmediatamente después de que IBM MQ classes for JMS haya generado un archivo FDC. Si esta variable se establece en TRUE, se genera un archivo de JavaCore en el directorio de trabajo del entorno de ejecución de Java en el que se ejecuta IBM MQ classes for JMS.

True
Generar archivo de JavaCore si el entorno de ejecución de Java está capacitado para hacerlo.

False
No generar archivo de JavaCore. Este es el valor predeterminado.

La stanza Properties de IBM MQ
Utilice la stanza Properties de IBM MQ para establecer las propiedades que afectan a la forma en que las clases IBM MQ classes for JMS interactúan con IBM MQ.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Cuando una aplicación que utiliza las clases IBM MQ classes for JMS se conecta a un gestor de colas de IBM MQ utilizando la modalidad de migración del proveedor de mensajería de IBM MQ, las IBM MQ classes for JMS utilizan un tamaño del almacenamiento intermedio predeterminado de 4 KB cuando la aplicación recibe mensajes. Si el mensaje que la aplicación intenta obtener es mayor que 4 KB, IBM MQ classes for JMS cambia el tamaño del almacenamiento intermedio para que sea lo suficientemente grande como para dar cabida al mensaje. El tamaño ampliado del almacenamiento intermedio se utiliza luego cuando se reciben los mensajes subsiguientes.

Esta propiedad controla cuándo el tamaño del almacenamiento intermedio se reduce de nuevo a 4 KB. De forma predeterminada, cuando se reciben diez mensajes consecutivos que son menores que el tamaño de almacenamiento intermedio más grande, el tamaño de almacenamiento intermedio se

reduce de nuevo a 4 KB. Para restablecer el tamaño del almacenamiento intermedio de nuevo en 4 KB cada vez que se recibe un mensaje, establezca la propiedad en el valor 0.

0

El almacenamiento intermedio se restablece siempre en el tamaño predeterminado.

10

Éste es el valor predeterminado. El almacenamiento intermedio se redimensionará después del décimo mensaje.

com.ibm.msg.client.wmq.receiveConversionCCSID

Cuando una aplicación que utiliza las clases IBM MQ classes for JMS se conecta a un gestor de colas de IBM MQ utilizando la modalidad normal del proveedor de mensajería de IBM MQ, se puede establecer la propiedad `receiveConversionCCSID` para alterar temporalmente el valor de CCSID predeterminado en la estructura MQMD que se utiliza para recibir mensajes del gestor de colas. De forma predeterminada, el MQMD contiene un campo de CCSID establecido en 1208, pero esto se puede cambiar si, por ejemplo, el gestor de colas no puede convertir mensajes a esta página de códigos.

Los valores válidos son cualquier número de CCSID válido o uno de los valores siguientes:

-1

Utilizar el valor predeterminado de la plataforma.

1208

Éste es el valor predeterminado.

Stanza specifics en modo cliente

La stanza specifics en modo cliente se usa para especificar propiedades que se usan cuando las IBM MQ classes for JMS se conectan con un gestor de colas que usa el transporte CLIENT.

com.ibm.mq.polling.RemoteRequestEntry

Especifica el intervalo de sondeo que utiliza IBM MQ classes for JMS para comprobar si hay conexiones interrumpidas cuando está esperando una respuesta de un gestor de colas.

Entero positivo

Número de milisegundos que se espera antes de comprobar. El valor predeterminado es de 10000, o 10 segundos. El valor mínimo es de 3000 y los valores inferiores se tratan de la misma forma que este valor mínimo.

Propiedades utilizadas para configurar el comportamiento del cliente JMS

Utilice estas propiedades para configurar el comportamiento del cliente JMS.

com.ibm.mq.jms.SupportMQExtensions VERDADERO|FALSO

La especificación JMS 2.0 introduce cambios en el funcionamiento de determinados comportamientos. IBM MQ 8.0 incluye la propiedad `com.ibm.mq.jms.SupportMQExtensions`, que puede establecerse en *TRUE* para revertir los comportamientos modificados a las implementaciones anteriores. La reversión de los comportamientos cambiados podría ser necesario para algunas aplicaciones JMS 2.0 y, también, para algunas aplicaciones que utilizan la API JMS 1.1, pero que ejecutan IBM MQ 8.0 IBM MQ classes for JMS.

TRUE

Las siguientes tres áreas de funcionalidad se revierten estableciendo SupportMQExtensions en TRUE:

Prioridad de mensaje

A los mensajes se les puede asignar una prioridad de *0* a *9*. Antes de JMS 2.0, los mensajes también podían utilizar el valor *-1*, lo que indica que se utiliza la prioridad predeterminada de una cola. JMS 2.0 no permite establecer la prioridad de mensaje en *-1*. La activación de `SupportMQExtensions` permite utilizar el valor *-1*.

ID de cliente
La especificación JMS 2.0 requiere que se compruebe la exclusividad de los ID de cliente no nulos cuando realizan una conexión. La activación de `SupportMQExtensions` significa que este requisito se descarta y el ID de cliente puede reutilizarse.

`NoLocal`
La especificación JMS 2.0 requiere que, cuando esta constante esté activada, un consumidor no pueda recibir los mensajes publicados por el mismo ID de cliente. Antes de JMS 2.0, este atributo estaba establecido en un suscriptor para evitar la recepción mensajes publicados por su propia conexión. La activación de `SupportMQExtensions` revierte este comportamiento a su implementación anterior.

FALSE
Los cambios de comportamiento se conservan.

`com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= VERDADERO|FALSO`
A partir de IBM MQ 8.0.0 Fix Pack 2, si una aplicación ha enviado un mensaje de bytes o secuencia, IBM MQ classes for JMS puede establecer el estado del mensaje que se acaba de enviar en solo lectura o solo escritura.

TRUE
Los objetos se establecen en solo lectura después de enviarse. El establecimiento de este valor mantiene la compatibilidad con la especificación JMS 2.0.

FALSE
Los objetos se establecen en solo escritura después de enviarse. Éste es el valor predeterminado.

Conceptos relacionados

“Propiedad SupportMQExtensions” en la página 337
La especificación JMS 2.0 ha introducido cambios en la forma en que funcionan determinados comportamientos. IBM MQ 8.0 y posteriores incluyen la propiedad **`com.ibm.mq.jms.SupportMQExtensions`**, que se puede establecer en TRUE para revertir estos comportamientos cambiados a implementaciones anteriores.

Configuración de STEPLIB para IBM MQ classes for JMS en z/OS
En z/OS, la biblioteca STEPLIB que se utiliza en tiempo de ejecución debe contener las bibliotecas SCSQAUTH y SCSQANLE de IBM MQ. Especifique estas bibliotecas en el JCL de inicio o mediante el archivo `.profile`.

A partir de z/OS UNIX System Services, puede añadirlos utilizando una línea en `.profile` tal como se muestra en el siguiente fragmento de código, sustituyendo `thlqual` por el calificador de conjunto de datos de alto nivel que ha elegido al instalar IBM MQ:

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

En otros entornos, normalmente es necesario editar el JCL de inicio para incluir SCSQAUTH y SCSQANLE en la concatenación STEPLIB:

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

IBM MQ classes for JMS y herramientas de gestión de software
Las herramientas de gestión de software como Apache Maven se pueden utilizar con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

Muchas organizaciones de desarrollo de gran tamaño utilizan estas herramientas para gestionar de forma centralizada los repositorios de bibliotecas de terceros.

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging se componen de una serie de archivos JAR. Cuando se desarrollan aplicaciones de lenguaje Java utilizando esta API, es necesaria una instalación de IBM MQ Server, IBM MQ Client o IBM MQ Client SupportPac en la máquina en la que se está desarrollando la aplicación.

Si desea utilizar una herramienta de este tipo y añadir los archivos JAR que conforman IBM MQ classes for JMS a un repositorio gestionado centralmente, se deben observar los puntos siguientes:

- Hay que poner un repositorio o contenedor a disposición de los desarrolladores de la organización únicamente. No se permite ninguna distribución fuera de la organización.
- El repositorio debe contener un conjunto completo y coherente de archivos JAR de un único release o fixpack de IBM MQ.
- Usted es responsable de actualizar el repositorio con cualquier mantenimiento proporcionado el equipo de soporte de IBM.

Es necesario instalar los siguientes archivos JAR en el repositorio:

- JMS 2.0 `com.ibm.mq.allclient.jar` y `jms.jar` son necesarios si está utilizando IBM MQ classes for JMS.
- JM 3.0 `com.ibm.mq.jakarta.client.jar` y `jakarta.jms-api.jar` son necesarios si utiliza IBM MQ classes for Jakarta Messaging.
- `fscontext.jar` es necesario si está utilizando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging y está accediendo a objetos administrados de JMS que están almacenados en un contexto JNDI de sistema de archivos.
- `providerutil.jar` es necesario si está utilizando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging y está accediendo a objetos administrados JMS que están almacenados en un contexto JNDI de sistema de archivos.
- El proveedor de seguridad de Bouncy Castle y los archivos JAR de soporte de CMS son necesarios para el soporte de los JRE que no son deIBM . Para obtener más información, consulte Soporte de los JRE no de IBM.

Ejecución de aplicaciones de IBM MQ classes for JMS en el Java security manager

IBM MQ classes for JMS puede ejecutarse con el Java security manager habilitado. Para ejecutar correctamente aplicaciones con el Java security manager habilitado, debe configurar la Java Virtual Machine (JVM) con un archivo de configuración de políticas adecuado.

La forma más sencilla de crear un archivo de definición de políticas adecuado es cambiar el archivo de configuración de políticas que se proporciona con el Java runtime environment (JRE). En la mayoría de los sistemas, este archivo se encuentra en el directorio `lib/security/java.policy` relativo al directorio JRE. Puede editar el archivo de configuración de políticas utilizando su editor preferido o el programa de herramienta de política que se proporciona con el JRE.

Archivo de configuración de políticas de ejemplo

A continuación, se muestra un ejemplo de un archivo de configuración de políticas que permite a IBM MQ classes for JMS ejecutarse correctamente con el gestor de seguridad predeterminado. Este archivo deberá personalizarse para especificar las ubicaciones de determinados archivos y directorios: *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que se ha instalado IBM MQ, *MQ_DATA_DIRECTORY* representa la ubicación del directorio de datos MQ y *QM_NAME* es el nombre del gestor de colas para el que se está configurando el acceso.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";
```

```
  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

  //For the client transport type.
  permission java.net.SocketPermission "*","connect,resolve";

  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";

  //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

  //For applications that use User Exits
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
  permission java.lang.RuntimePermission "createClassLoader";

  //Required for the z/OS platform
  permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

  // Used by the internal ConnectionFactory implementation
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

  // Used for controlled class loading
  permission java.lang.RuntimePermission "setContextClassLoader";

  // Used to default the Application name in Client mode connections
  permission java.util.PropertyPermission "sun.java.command","read";

  // Used by the IBM JSSE classes
  permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

  //Required to determine if an IBM Java Runtime is running in FIPS mode,
  //and to modify the property values status as required.
  permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
  permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
  //Required if an IBM FIPS provider is to be used for SSL communication.
  permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

  // Required for non-IBM Java Runtimes that establish secure client
  // transport mode connections using mutual TLS authentication
  permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
  permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

En el ejemplo, la sentencia `grant` contiene los permisos necesarios para IBM MQ classes for JMS. Para utilizar estas sentencias grant en el archivo de configuración de política, es posible que necesite modificar los nombres de la vía de acceso en función de dónde ha instalado IBM MQ classes for JMS y dónde almacena las aplicaciones.

Las aplicaciones de ejemplo que se suministran con IBM MQ classes for JMS y los scripts para ejecutarlas no habilitan al gestor de seguridad.

Importante:

El recurso de rastreo IBM MQ classes for JMS requiere más permisos cuando realiza consultas adicionales de propiedades del sistema y también cuando realiza más operaciones del sistema de archivos.

En el directorio `samples/wmqjava` de la instalación de IBM MQ, se proporciona un archivo de política de seguridad de plantilla adecuado para que se ejecute bajo un gestor de seguridad con el rastreo habilitado como `example.security.policy`.

Configuración posterior a la instalación para aplicaciones de IBM MQ classes for JMS

Este tema describe qué autorizaciones necesitan las aplicaciones de IBM MQ classes for JMS para acceder a los recursos de un gestor de colas. También describe modalidades de conexión y cómo configurar un gestor de colas para que las aplicaciones se puedan conectar en la modalidad de cliente.

Recuerde examinar el archivo léame de IBM MQ. Podría contener información que reemplace la información de este tema.

Objetos utilizados por JMS cuyo acceso por usuarios no privilegiados requiere autorización

Los usuarios no privilegiados necesitan recibir autorización para acceder a las colas utilizadas por JMS. Todas las aplicaciones de JMS necesitan autorización para acceder el gestor de colas con el cual trabajan.

Para obtener detalles sobre el control de accesos en IBM MQ, consulte Configuración de la seguridad.

Las aplicaciones de IBM MQ classes for JMS necesitan autorización para `connect` y para `inq` con respecto al gestor de colas. Puede establecer las autorizaciones adecuadas utilizando el mandato de control **`setmqaut`**, por ejemplo:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Para el dominio punto a punto, son necesarias las autorizaciones siguientes:

- Las colas que son utilizadas por los objetos MessageProducer necesitan autorización para `put`.
- Las colas que son utilizadas por los objetos MessageConsumer y QueueBrowser necesitan autorizaciones para `get`, `inq` y `browse`.
- El método QueueSession.createTemporaryQueue () necesita acceso a la cola de modelo especificada por la propiedad TEMPMODEL del objeto QueueConnectionFactory. De forma predeterminada, esta cola de modelo es SYSTEM.TEMP.MODEL.QUEUE.

Si cualquiera de estas colas es una cola alias, sus colas de destino necesitan autorización para inquire. Si la cola de destino es una cola de clúster, también necesita autorización para browse.

Para el dominio de publicación/suscripción, se utilizan las colas siguientes si las clases IBM MQ classes for JMS se conectan a un gestor de colas de IBM MQ en la modalidad de migración del proveedor de mensajería de IBM MQ:

- 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

Para obtener más información sobre la modalidad de migración del proveedor de mensajería IBM MQ, consulte Configuración de la propiedad JMS **PROVIDERVERSION**

Además, si las IBM MQ classes for JMS se conectan a un gestor de colas en esta modalidad, cualquier aplicación que publique mensajes necesita acceso a la cola de corriente especificada por el objeto TopicConnectionFactory o el objeto de tema. De forma predeterminada, esta cola es SYSTEM.BROKER.DEFAULT.STREAM.

Si utiliza ConnectionConsumer, el adaptador de recursos IBM MQ o el proveedor de mensajería WebSphere Application Server IBM MQ, podría ser necesaria una autorización adicional.

Las colas que deben ser leídas por ConnectionConsumer deben tener autorizaciones para `get`, `inq` y `browse`. La cola de mensajes no entregados del sistema y cualquier cola de retirada o cola de informe utilizada por ConnectionConsumer debe tener autorizaciones para `put` y `passall`.

Cuando una aplicación utiliza la modalidad normal del proveedor de mensajería de IBM MQ para realizar la mensajería de publicación/suscripción, la aplicación utiliza la funcionalidad de publicación/suscripción integrada que proporciona el gestor de colas. Consulte Seguridad de publicación/suscripción para obtener información sobre cómo proteger los temas y las colas que se utilizan.

Modalidades de conexión para IBM MQ classes for JMS
Una aplicación de IBM MQ classes for JMS se puede conectar a un gestor de colas en la modalidad de cliente o en la modalidad de enlaces. En la modalidad de cliente, IBM MQ classes for JMS se conecta al gestor de colas a través de TCP/IP. En la modalidad de enlaces, IBM MQ classes for JMS se conecta directamente al gestor de colas utilizando la interfaz nativa de Java (JNI) .

Una aplicación que se ejecuta en WebSphere Application Server en z/OS puede conectarse a un gestor de colas en modalidad de enlaces o de cliente, pero una aplicación que se ejecuta en cualquier otro entorno en z/OS sólo puede conectarse a un gestor de colas en modalidad de enlaces. Una aplicación que se ejecuta en cualquier plataforma se puede conectar a un gestor de colas en modalidad de enlaces o de cliente.

Puede utilizar la versión actual o cualquier versión soportada anterior de IBM MQ classes for JMS con un gestor de colas actual, y puede utilizar una versión soportada actual o anterior del gestor de colas con la versión actual de IBM MQ classes for JMS. Si mezcla versiones diferentes, la función está limitada al nivel de la versión más anterior.

Las secciones siguientes describen cada modalidad de conexión con más detalle.

Modalidad de cliente

Para conectar con un gestor de colas en la modalidad de cliente, una aplicación de IBM MQ classes for JMS se puede ejecutar en el mismo sistema en el que se ejecuta el gestor de colas, o en un sistema distinto. En cada caso, IBM MQ classes for JMS se conecta con el gestor de colas a través de TCP/IP.

Modalidad de enlaces

Para conectar con un gestor de colas en la modalidad de enlaces, una aplicación de IBM MQ classes for JMS debe ejecutarse en el mismo sistema en el que se está ejecutando el gestor de colas.

IBM MQ classes for JMS se conecta directamente al gestor de colas utilizando la interfaz nativa de Java (JNI). Para utilizar el transporte de enlaces, IBM MQ classes for JMS se debe ejecutar en un entorno que tenga acceso a las bibliotecas de la interfaz nativa de IBM MQ Java; consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 100 para obtener más información.

Las IBM MQ classes for JMS soportan los valores siguientes para *ConnectOption*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Para cambiar las opciones de conexión utilizadas por las IBM MQ classes for JMS, modifique la propiedad CONNOPT de la Fábrica de conexiones.

Para obtener más información sobre las opciones de conexión, consulte “Conexión a un gestor de colas mediante la llamada MQCONNX” en la página 756

Para utilizar el transporte de enlaces, el entorno de ejecución de Java que se utiliza debe soportar el CCSID (identificador de juego de caracteres codificados) del gestor de colas al que se conecta IBM MQ classes for JMS.

Los detalles sobre cómo determinar qué CCSID están soportados por un Java Runtime Environment se pueden encontrar en IBM MQ FDC con el ID de analizador 21 generado al utilizar clases IBM MQ V7 Java o IBM MQ V7 para JMS.

Configuración del gestor de colas para que las aplicaciones de IBM MQ classes for JMS se puedan conectar en la modalidad de cliente
Para configurar el gestor de colas de modo que las aplicaciones de IBM MQ classes for JMS se puedan conectar en la modalidad de cliente, debe crear una definición de canal de conexión de servidor e iniciar un proceso de escucha.

Creación de una definición de canal de conexión con el servidor

En todas la plataformas, puede utilizar el mandato MQSC DEFINE CHANNEL para crear una definición de canal de conexión de servidor. Vea el ejemplo siguiente:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i En IBM i, como alternativa puede utilizar el mandato de CL CRTMQMCHL, como en el ejemplo siguiente:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

En este mandato, *NOMBGSTCOLAS* es el nombre del gestor de colas.

Windows Linux En Linux y Windows, puede también crear una definición de canal de conexión de servidor utilizando IBM MQ Explorer.

z/OS En z/OS, puede utilizar los paneles de operaciones y de control para crear una definición de canal de conexión de servidor.

El nombre del canal (JAVA.CHANNEL en los ejemplos anteriores) debe ser el mismo que el nombre del canal especificado por la propiedad CHANNEL de la fábrica de conexiones que la aplicación utiliza para conectarse al gestor de colas. El valor predeterminado de la propiedad CHANNEL es SYSTEM.DEF.SVRCONN.

Inicio de un escucha

Debe iniciar un escucha para el gestor de colas si todavía no ha iniciado uno.

Multi En Multiplatforms, puede utilizar el mandato de MQSC START LISTENER para iniciar un proceso de escucha después de crear primero un objeto de escucha mediante el mandato de MQSC DEFINE LISTENER, tal como se muestra en el ejemplo siguiente:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS En z/OS, utilice sólo el mandato START LISTENER, como en el ejemplo siguiente, pero tenga en cuenta que el espacio de direcciones del iniciador de canal debe iniciarse antes de poder iniciar un proceso de escucha:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i En IBM i, también puede utilizar el mandato de CL STRMQMLSR para iniciar un proceso de escucha, tal como en el ejemplo siguiente:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

En este mandato, *NOMBGSTCOLAS* es el nombre del gestor de colas.

ALW En AIX, Linux, and Windows, también puede utilizar el mandato de control **runmqlsr** para iniciar un escucha, como en el ejemplo siguiente:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

En este mandato, *NombGstColas* es el nombre del gestor de colas.

Windows Linux En Linux y Windows, puede también iniciar un proceso de escucha mediante IBM MQ Explorer.

z/OS En z/OS, puede también utilizar los paneles de operaciones y de control para iniciar un proceso de escucha.

El número del puerto donde se está a la escucha debe ser el mismo que el número de puerto especificado por la propiedad PORT de la fábrica de conexiones que la aplicación utiliza para conectar con el gestor de colas. El valor predeterminado de la propiedad PORT es 1414.

Prueba de verificación de la instalación punto a punto para IBM MQ classes for JMS

Se proporciona un programa de prueba de verificación de instalación punto a punto (IVT) con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging. El programa se conecta a un gestor de colas en modalidad de enlaces o de cliente, envía un mensaje a la cola denominada SYSTEM.DEFAULT.LOCAL.QUEUE y recibe el mensaje de la cola. El programa puede crear y configurar todos los objetos que requiere de forma dinámica en el tiempo de ejecución, o bien puede utilizar JNDI para recuperar objetos administrados de un servicio de directorio.

Ejecute la prueba de verificación de la instalación sin utilizar primero JNDI, pues la prueba es autónoma y no necesita el uso de un servicio de directorio. Para obtener una descripción de los objetos administrados, consulte Configuración de objetos de JMS utilizando la herramienta de administración.

Prueba de verificación de la instalación punto a punto sin utilizar JNDI

En esta prueba, el programa IVT crea y configura dinámicamente todos los objetos que necesita en tiempo de ejecución y no utiliza JNDI.

Multi En multiplataformas, se proporciona un script para ejecutar el programa IVT. El script se denomina **IVTRun** en sistemas AIX and Linux y **IVTRun.bat** en Windows. el script se encuentra en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS . La vía de acceso de clases debe contener `com.ibm.mqjms.jar`.

Para ejecutar la prueba en la modalidad de enlaces, emita el mandato siguiente:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Para ejecutar la prueba en modalidad de cliente, primero configure el gestor de colas tal como se describe en “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097. Tenga en cuenta que el canal que se va a utilizar toma como valor predeterminado **SYSTEM.DEF.SVRCONN** y la cola que se va a utilizar es **SYSTEM.DEFAULT.LOCAL.QUEUE**y, a continuación, especifique el mandato siguiente:

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

z/OS No se proporciona ningún script equivalente en los sistemas z/OS . En su lugar, ejecute la IVT en modalidad de enlaces invocando la clase Java directamente. En z/OS, puede elegir entre dos instancias funcionalmente idénticas del programa IVT:

- `com.ibm.mq.jms.MQJMSIVT`, que está disponible con IBM MQ classes for JMS (JMS 2.0). Para utilizar este programa, la vía de acceso de clases debe contener `com.ibm.mqjms.jar` o `com.ibm.mq.allclient.jar`.
- `com.ibm.mq.jakarta.jms.MQJMSIVT`, que está disponible con IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Para utilizar este programa, la vía de acceso de clases debe contener `com.ibm.mq.jakarta.client.jar`.

Para ejecutar la prueba en modalidad de enlaces en z/OS, especifique el mandato siguiente:

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

Los parámetros contenidos en los mandatos tienen los significados siguientes:

-m *gstc*
Nombre del gestor de colas al que se conecta el programa IVT. Si ejecuta la prueba en la modalidad de enlaces y omite este parámetro, el programa IVT se conecta al gestor de colas predeterminado.

-host *nombreHost*
Nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.

-port *puerto*
Número del puerto en el que el escucha del gestor de colas está a la escucha. El valor predeterminado es 1414.

-channel *canal*
Nombre del canal MQI que el programa IVT utiliza para conectarse al gestor de colas. El valor predeterminado es `SYSTEM.DEF.SVRCONN`.

-v *versión_proveedor*
Nivel de release del gestor de colas al que se debe conectar el programa IVT.

Este parámetro se utiliza para establecer la propiedad PROVIDERVERSION de un objeto MQQueueConnectionFactory y tiene los mismos valores válidos que los de la propiedad PROVIDERVERSION. Por lo tanto, para obtener más información sobre este parámetro, incluidos sus valores válidos, consulte JMS: cambios en la propiedad PROVIDERVERSION y la descripción de la propiedad PROVIDERVERSION en Propiedades de objetos IBM MQ classes for JMS.

El valor predeterminado es `unspecified`.

-ccsid *ccsid*
Identificador (CCSID) del juego de caracteres codificado o página de códigos que utilizará la conexión. El valor predeterminado es 819.

-t
El rastreo está habilitado. De forma predeterminada, el rastreo está inhabilitado.

Una prueba satisfactoria produce una salida similar a la salida de ejemplo siguiente:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again
```

```
Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Prueba de verificación de la instalación punto a punto utilizando JNDI

Multi

En esta prueba, el programa IVT utiliza JNDI para recuperar objetos administrados de un servicio de directorio.

Para poder ejecutar la prueba, debe configurar un servicio de directorio que esté basado en un servidor Lightweight Directory Access Protocol (LDAP) o el sistema de archivos local. También debe configurar la herramienta de administración de IBM MQ JMS para que pueda utilizar el servicio de directorio para almacenar objetos administrados. Para obtener más información sobre estos requisitos previos, consulte “Requisitos previos para IBM MQ classes for JMS” en la página 91. Para obtener más información sobre cómo configurar la herramienta de administración de IBM MQ JMS, consulte Configuración de la herramienta de administración de JMS.

El programa IVT debe poder utilizar JNDI para recuperar un objeto MQQueueConnectionFactory y un objeto MQQueue del servicio de directorio. Se proporciona un script para crear estos objetos administrados automáticamente. El script se denomina IVTSetup en los sistemas AIX and Linux, e IVTSetup.bat en Windows, y reside en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS. Para ejecutar el script, entre el mandato siguiente:

```
IVTSetup
```

El script invoca la herramienta de administración de IBM MQ JMS para crear los objetos administrados.

El objeto MQQueueConnectionFactory está vinculado al nombre ivtQCF y se crea con los valores predeterminados para todas las propiedades, lo que significa que el programa IVT se ejecuta en la modalidad de enlaces y se conecta al gestor de colas predeterminado. Si desea que el programa IVT se ejecute en la modalidad de cliente, o que se conecte a un gestor de colas distinto al gestor de colas predeterminado, debe utilizar la herramienta de administración de IBM MQ JMS o IBM MQ Explorer para cambiar las propiedades apropiadas del objeto MQQueueConnectionFactory. Para obtener más información sobre cómo utilizar la herramienta de administración de IBM MQ Explorer JMS, consulte Configuración de objetos JMS utilizando la herramienta de administración. Para obtener más información

sobre cómo utilizar IBM MQ Explorer, consulte Introducción a IBM MQ Explorer o la ayuda proporcionada con IBM MQ Explorer.

El objeto MQQueue está enlazado con el nombre ivtQ y se crea con los valores predeterminados para todas sus propiedades, excepto para la propiedad QUEUE, cuyo valor es SYSTEM.DEFAULT.LOCAL.QUEUE.

Cuando haya creado los objetos administrados, puede ejecutar el programa IVT. Para ejecutar la prueba utilizando JNDI, entre el mandato siguiente:

```
IVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Los parámetros contenidos en el mandato tienen los significados siguientes:

-url "*URL_proveedor*"
El localizador uniforme de recursos (URL) del servicio de directorio. El URL puede tener uno de los formatos siguientes:

- `ldap://hostname/contextName` , para un servicio de directorio basado en un servidor LDAP
- `file:/directoryPath` , para un servicio de directorio basado en el sistema de archivos local

Observe que debe encerrar el URL entre comillas (").

-icf *fábrica_contexto_inicial*
Nombre de clase de la fábrica de contexto inicial, que debe ser uno de los valores siguientes:

- `com.sun.jndi.ldap.LdapCtxFactory`, para un servicio de directorio basado en un servidor LDAP. Éste es el valor predeterminado.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para un servicio de directorio basado en el sistema de archivos local.

-t
El rastreo está habilitado. De forma predeterminada, el rastreo está inhabilitado.

Una prueba satisfactoria produce una salida similar a la de la prueba satisfactoria sin JNDI. La diferencia principal es que la salida indica que la prueba está utilizando JNDI para recuperar un objeto MQQueueConnectionFactory y un objeto MQQueue.

Si bien no es estrictamente necesario, se aconseja efectuar una limpieza después de la prueba suprimiendo los objetos administrados que ha creado el script IVTSetup. Para esta finalidad, se suministra un script. El script se denomina IVTTidy en los sistemas AIX and Linux e IVTTidy.bat en Windows, y reside en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS.

Determinación de problemas para la prueba de verificación de la instalación punto a punto

La prueba de verificación de la instalación puede fallar por las razones siguientes:

- Si el programa IVT escribe un mensaje que indica que no puede encontrar una clase, compruebe si la vía de acceso de clases se ha establecido correctamente, tal como se describe en la “Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging” en la página 97.
- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

  ```
  Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
  and host name ' hostname '
  ```

  y un código de razón asociado de 2059. Las variables contenidas en el mensaje tienen los significados siguientes:

gestor_colas>
Nombre del gestor de colas al que se está intentando conectar el programa IVT. Esta inserción de mensaje está en blanco si el programa IVT está intentando conectarse al gestor de colas predeterminado en la modalidad de enlaces.

connMode
La modalidad de conexión, que puede ser `Bindings` o `Client`.

Nombre de host
Nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.

Este mensaje significa que el gestor de colas al que el programa IVT se está intentando conectar no está disponible. Compruebe si el gestor de colas está en ejecución y, si el programa IVT está intentando conectarse al gestor de colas predeterminado, asegúrese de que el gestor de colas está definido como el gestor de colas predeterminado del sistema.

- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

Este mensaje significa que la cola SYSTEM.DEFAULT.LOCAL.QUEUE no existe en el gestor de colas al que el programa IVT está conectado. O bien, si la cola existe, el programa IVT no puede abrir la cola porque no está habilitado para transferir y recibir mensajes. Compruebe si la cola existe y si está habilitada para transferir y recibir mensajes.

- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
Unable to bind to object
```

Este mensaje significa que no existe una conexión con el servidor LDAP, pero que el servidor LDAP no está configurado correctamente. El servidor LDAP no está configurado para almacenar objetos Java o los permisos sobre los objetos o el sufijo no son correctos. Si desea más ayuda para esta situación, consulte la documentación del servidor LDAP.

- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

Este mensaje significa que el gestor de colas no se ha configurado correctamente para aceptar una conexión de cliente desde el sistema. Consulte “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097 para obtener detalles.

Prueba de verificación de la instalación de publicación/suscripción para IBM MQ classes for JMS

Se proporciona un programa de prueba de verificación de instalación de publicación/suscripción con IBM MQ classes for JMS. El programa se conecta a un gestor de colas en modalidad de enlaces o de cliente, se suscribe a un tema, publica un mensaje sobre el tema y luego recibe el mensaje que acaba de publicar. El programa puede crear y configurar todos los objetos que requiere de forma dinámica en el tiempo de ejecución, o bien puede utilizar JNDI para recuperar objetos administrados de un servicio de directorio.

Ejecute la prueba de verificación de la instalación sin utilizar primero JNDI, pues la prueba es autónoma y no necesita el uso de un servicio de directorio. Para obtener una descripción de los objetos administrados, consulte Configuración de objetos de JMS utilizando la herramienta de administración.

Prueba de verificación de la instalación de publicación/suscripción sin utilizar JNDI

En esta prueba, el programa IVT crea y configura dinámicamente todos los objetos que necesita en tiempo de ejecución y no utiliza JNDI.

Se suministra un script para ejecutar el programa IVT. El script se denomina PSIVTRun en los sistemas AIX and Linux e PSIVTRun.bat en Windows, y reside en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS.

Para ejecutar la prueba en la modalidad de enlaces, emita el mandato siguiente:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Para ejecutar la prueba en la modalidad de cliente, primero configure el gestor de colas tal como se describe en "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097 teniendo en cuenta que el canal que se debe utilizar se establece de forma predeterminada en SYSTEM.DEF.SVRCONN y emita el mandato siguiente:

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

Los parámetros contenidos en los mandatos tienen los significados siguientes:

-m *gstc*
Nombre del gestor de colas al que se conecta el programa IVT. Si ejecuta la prueba en la modalidad de enlaces y omite este parámetro, el programa IVT se conecta al gestor de colas predeterminado.

-host *nombreHost*
Nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.

-port *puerto*
Número del puerto en el que el escucha del gestor de colas está a la escucha. El valor predeterminado es 1414.

-channel *canal*
Nombre del canal MQI que el programa IVT utiliza para conectarse al gestor de colas. El valor predeterminado es `SYSTEM.DEF.SVRCONN`.

-bqm *gestor_colas_intermediario*
Nombre del gestor de colas en el que se ejecuta el intermediario. El valor predeterminado es el nombre del gestor de colas al que se conecta el programa IVT.

Este parámetro no es aplicable si el número de versión del gestor de colas v es 7 o mayor.

-v *versión_proveedor*
Nivel de release del gestor de colas al que se debe conectar el programa IVT.

Este parámetro se utiliza para establecer la propiedad PROVIDERVERSION de un objeto MQTopicConnectionFactory y tiene los mismos valores válidos que los de la propiedad PROVIDERVERSION. Por lo tanto, para obtener más información sobre este parámetro, incluidos sus valores válidos, consulte la descripción de la propiedad PROVIDERVERSION en Propiedades de objetos IBM MQ classes for JMS.

El valor predeterminado es `unspecified`.

-ccsid *ccsid*
Identificador (CCSID) del juego de caracteres codificado o página de códigos que utilizará la conexión. El valor predeterminado es 819.

-t
El rastreo está habilitado. De forma predeterminada, el rastreo está inhabilitado.

Una prueba satisfactoria produce una salida similar a la salida de ejemplo siguiente:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
```

```
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

Prueba de verificación de la instalación de de publicación/suscripción utilizando JNDI

En esta prueba, el programa IVT utiliza JNDI para recuperar objetos administrados de un servicio de directorio.

Para poder ejecutar la prueba, debe configurar un servicio de directorio que esté basado en un servidor Lightweight Directory Access Protocol (LDAP) o el sistema de archivos local. También debe configurar la herramienta de administración de IBM MQ JMS para que pueda utilizar el servicio de directorio para almacenar objetos administrados. Para obtener más información sobre estos requisitos previos, consulte “Requisitos previos para IBM MQ classes for JMS” en la página 91. Para obtener más información sobre cómo configurar la herramienta de administración de IBM MQ JMS, consulte Configuración de la herramienta de administración de JMS.

El programa IVT debe poder utilizar JNDI para recuperar un objeto MQTopicConnectionFactory y un objeto MQTopic del servicio de directorio. Se proporciona un script para crear estos objetos administrados automáticamente. El script se denomina IVTSetup en los sistemas AIX and Linux, e IVTSetup.bat en Windows, y reside en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS. Para ejecutar el script, entre el mandato siguiente:

```
IVTSetup
```

El script invoca la herramienta de administración de IBM MQ JMS para crear los objetos administrados.

El objeto MQTopicConnectionFactory está vinculado con el nombre ivtTCF y se crea con los valores predeterminados para todas las propiedades, lo que significa que el programa IVT se ejecuta en modalidad de enlaces, se conecta al gestor de colas predeterminado y utiliza la función de publicación/

suscripción incluida. Si desea que el programa IVT se ejecute en la modalidad cliente, conéctese a un gestor de colas distinto al gestor de colas predeterminado, o utilice IBM Integration Bus en lugar de la función publicación/suscripción incorporada, debe utilizar la herramienta de administración de IBM MQ JMS o IBM MQ Explorer para cambiar las propiedades apropiadas del objeto MQTopicConnectionFactory. Para obtener más información sobre cómo utilizar la herramienta de administración de IBM MQ JMS, consulte Configuración de objetos JMS utilizando la herramienta de administración. Para obtener información sobre cómo utilizar IBM MQ Explorer, consulte la ayuda que se proporciona con IBM MQ Explorer.

El objeto MQTopic está enlazado con el nombre ivtT y se crea con los valores predeterminados para todas sus propiedades, excepto para la propiedad TOPIC, que tiene el valor MQJMS/PSIVT/Information.

Cuando haya creado los objetos administrados, puede ejecutar el programa IVT. Para ejecutar la prueba utilizando JNDI, entre el mandato siguiente:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Los parámetros contenidos en el mandato tienen los significados siguientes:

-url "*URL_proveedor*"
El localizador uniforme de recursos (URL) del servicio de directorio. El URL puede tener uno de los formatos siguientes:

- `ldap://hostname/contextName` , para un servicio de directorio basado en un servidor LDAP
- `file:/directoryPath` , para un servicio de directorio basado en el sistema de archivos local

Observe que debe encerrar el URL entre comillas (").

-icf *fábrica_contexto_inicial*
Nombre de clase de la fábrica de contexto inicial, que debe ser uno de los valores siguientes:

- `com.sun.jndi.ldap.LdapCtxFactory`, para un servicio de directorio basado en un servidor LDAP. Éste es el valor predeterminado.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para un servicio de directorio basado en el sistema de archivos local.

-t
El rastreo está habilitado. De forma predeterminada, el rastreo está inhabilitado.

Una prueba satisfactoria produce una salida similar a la de la prueba satisfactoria sin JNDI. La principal diferencia radica en que la salida indica que la prueba está utilizando JNDI para recuperar un objeto MQTopicConnectionFactory y un objeto MQTopic.

Si bien no es estrictamente necesario, se aconseja efectuar una limpieza después de la prueba suprimiendo los objetos administrados que ha creado el script IVTSetup. Para esta finalidad, se suministra un script. El script se denomina IVTTidy en los sistemas AIX and Linux e IVTTidy.bat en Windows, y reside en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS.

Determinación de problemas de la prueba de verificación de la instalación de publicación/suscripción

La prueba de verificación de la instalación puede fallar por las razones siguientes:

- Si el programa IVT escribe un mensaje que indica que no puede encontrar una clase, compruebe si la vía de acceso de clases se ha establecido correctamente, tal como se describe en la “Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging” en la página 97.
- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

y un código de razón asociado de 2059. Las variables contenidas en el mensaje tienen los significados siguientes:

gestor_colas>
Nombre del gestor de colas al que se está intentando conectar el programa IVT. Esta inserción de mensaje está en blanco si el programa IVT está intentando conectarse al gestor de colas predeterminado en la modalidad de enlaces.

connMode
La modalidad de conexión, que puede ser `Bindings` o `Client`.

Nombre de host
Nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.

Este mensaje significa que el gestor de colas al que el programa IVT se está intentando conectar no está disponible. Compruebe si el gestor de colas está en ejecución y, si el programa IVT está intentando conectarse al gestor de colas predeterminado, asegúrese de que el gestor de colas está definido como el gestor de colas predeterminado del sistema.

- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
Unable to bind to object
```

Este mensaje significa que no existe una conexión con el servidor LDAP, pero que el servidor LDAP no está configurado correctamente. El servidor LDAP no está configurado para almacenar objetos Java o los permisos sobre los objetos o el sufijo no son correctos. Si desea más ayuda para esta situación, consulte la documentación del servidor LDAP.

- Es posible que la prueba no se ejecute correctamente con el mensaje siguiente:

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

Este mensaje significa que el gestor de colas no está configurado correctamente para aceptar una conexión de cliente del sistema. Para obtener más información, consulte “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097.

JMS 2.0 *Utilización de las aplicaciones de ejemplo IBM MQ classes for JMS*

Las aplicaciones de ejemplo IBM MQ classes for JMS proporcionan una descripción general de las características comunes de la API JMS. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Acerca de esta tarea

Si necesita ayuda para crear sus propias aplicaciones, puede utilizar las aplicaciones de ejemplo como punto de partida. Se proporcionan ambas versiones, la de origen y la compilada, para cada aplicación. Revise el código fuente de ejemplo e identifique los pasos clave para crear cada objeto necesario para la aplicación (ConnectionFactory, Connection, Session, Destination, y un Producer, or un Consumer, o ambos), y para establecer cualquier propiedad específica necesaria para especificar cómo desea que funcione la aplicación. Para obtener más información, consulte “Escritura de aplicaciones IBM MQ classes for JMS/Jakarta Messaging” en la página 145. Los ejemplos podrían estar sujetos a cambios en futuros releases de IBM MQ.

Para JMS 2.0, Tabla 11 en la página 125 muestra dónde se instalan las aplicaciones de ejemplo de IBM MQ classes for JMS en cada plataforma.

Nota:

JM 3.0 Para IBM MQ classes for Jakarta Messaging, se están preparando nuevos ejemplos.

Tabla 11. Directorios de instalación para las aplicaciones de ejemplo IBM MQ classes for JMS	
Plataforma	**Directorio**
AIX Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

Dentro de este directorio, hay subdirectorios que contienen una o más aplicaciones de ejemplo, tal como se indica en Tabla 12 en la página 125.

Tabla 12. IBM MQ classes for JMS aplicaciones de ejemplo	
Nombre de ejemplo	**Descripción**
JmsBrowser.java	Una aplicación de navegador de colas JMS que consulta todos los mensajes disponibles en la cola determinada, sin eliminarlos, en el orden en el que los recibiría una aplicación de consumidor.
JmsConsumer.java	Una aplicación de navegador de colas JMS que consulta todos los mensajes disponibles en la cola especificada, sin eliminarlos, en el orden en el que los recibiría una aplicación de consumidor, buscando la instancia de la fábrica de conexiones y la instancia de destino en un contexto inicial (Este ejemplo solo da soporte al contexto del sistema de archivos).
JmsJndiConsumer.java	Una aplicación de consumidor JMS (receptor o suscriptor) que recibe un mensaje del destino determinado (cola o tema) buscando la instancia de la fábrica de conexiones y la instancia de destino en un contexto inicial (Este ejemplo solo da soporte al contexto del sistema de archivos).
JmsJndiProducer.java	Una aplicación de productor JMS (emisor o publicador) que envía un mensaje simple al destino determinado (cola o tema) buscando la instancia de la fábrica de conexiones y la instancia de destino en un contexto inicial (Este ejemplo solo da soporte al contexto del sistema de archivos).
JmsProducer.java	Una aplicación de productor JMS (emisor o publicador) que envía un mensaje simple al destino determinado (cola o tema).
/interactive/	
SampleConsumerJava.java	Recibir mensajes de un tema/cola.
SampleProducerJava.java	Enviar mensajes a un tema/cola.
/interactive/helper/	
BaseOptions.java	Clase abstracta que se puede ampliar para proporcionar la funcionalidad de opciones de usuario.
IsValidType.java	Clase abstracta para las clases de comprobador de validez.

Tabla 12. IBM MQ classes for JMS aplicaciones de ejemplo (continuación)	
Nombre de ejemplo	**Descripción**
JmsApp.java	Clase abstracta que se puede ampliar para proporcionar la funcionalidad de consumidor/productor.
Keys.java	Conjunto de claves que definen opciones para las aplicaciones de ejemplo.
Literals.java	Conjunto de literales de constante.
MyContext.java	Contexto en el que se presentan las opciones.
Options.java	Proporciona la funcionalidad para las opciones de usuario.
OptionsPresenter.java	Contexto en el cual se presentan las opciones actuales.
/simple/	
SimpleAsyncPutPTP.java	Una aplicación simple para la mensajería punto a punto; el mensaje se envía de forma asíncrona (también se conoce como mensajería *activar-y-olvidar*). No se recibe ningún mensaje.
SimpleDurableSub.java	Una aplicación simple que ilustra la prestación de suscripción duradera.
SimpleJNDILookup.java	Una aplicación mínima y simple que ilustra la búsqueda de objetos JMS utilizando el contexto inicial. No se realiza ninguna conexión al gestor de colas y no se envía ni recibe ningún mensaje.
SimpleMQMDRead.java	Una aplicación simple que ilustra cómo una aplicación JMS puede servir a los campos de MQ Message Descriptor (MQMD) como propiedades de mensaje JMS. No se envía ningún mensaje; se da por supuesto que la cola en uso se llena con algunos mensajes.
SimpleMQMDWrite.java	Una aplicación simple que ilustra cómo una aplicación JMS puede escribir campos de MQ Message Descriptor (MQMD). No se recibe ningún mensaje.
SimplePTP.java	Una aplicación mínima y simple para las mensajería punto a punto.
SimplePubSub.java	Una aplicación mínima y simple para la mensajería de publicación/suscripción.
SimpleReadAheadPTP.java	Una aplicación simple para la mensajería punto a punto; los mensajes se direccionan de forma continua desde el gestor de colas (también se conoce como recurso de lectura anticipada). No se envía ningún mensaje; se da por supuesto que la cola en uso se llena con algunos mensajes.
SimpleRequestor.java	Una aplicación simple que utiliza un solicitante para enviar un mensaje de solicitud y, después, esperar y recibir la respuesta. Nota: se da por supuesto que alguna otra aplicación procesará el mensaje de solicitud y enviará el mensaje de respuesta.
SimpleResponder.java	Una aplicación simple que está a la escucha en un destino para un mensaje y, después, envía una respuesta al destino replyTo del mensaje. La aplicación se escribe para que funcione junto con el ejemplo SimpleRequestor.
SimpleRetainedPub.java	Una aplicación simple que ilustra una publicación retenida. No se recibe ningún mensaje.
SimpleWMQJMSPTP.java	Una aplicación mínima y simple para las mensajería punto a punto.

Tabla 12. IBM MQ classes for JMS aplicaciones de ejemplo (continuación)	
Nombre de ejemplo	**Descripción**
SimpleWMQ JMSPubSub .java	Una aplicación mínima y simple para la mensajería de publicación/suscripción.

IBM MQ classes for JMS proporcionan un script llamado `runjms` que se puede utilizar para ejecutar las aplicaciones de ejemplo. Este script configura el entorno IBM MQ para permitirle ejecutar las aplicaciones de ejemplo IBM MQ classes for JMS.

Tabla 13 en la página 127 muestra la ubicación del script en cada plataforma:

Tabla 13. Ubicación del script `runjms`	
Plataforma	**Directorio**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	o
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Para utilizar el script `runjms` para invocar una aplicación de ejemplo, complete los pasos siguientes:

Procedimiento

1. Abra un indicador de mandatos y vaya hasta el directorio que contiene la aplicación de ejemplo que desea ejecutar.
2. Escriba el mandato siguiente:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   La aplicación de ejemplo muestra una lista de los parámetros que necesita.
3. Especifique el mandato siguiente para ejecutar el ejemplo con estos parámetros:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Ejemplo

Linux Por ejemplo, para ejecutar el ejemplo `JmsBrowser` en Linux, especifique los mandatos siguientes:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Conceptos relacionados

"¿Qué se instala para IBM MQ classes for JMS?" en la página 92
Se crean varios archivos y directorios cuando se instala IBM MQ classes for JMS. En Windows, se realizan algunas configuraciones durante la instalación estableciendo automáticamente variables de entorno. En

otras plataformas, y en algunos entornos Windows, debe establecer variables de entorno para poder ejecutar aplicaciones de IBM MQ classes for JMS.

Scripts proporcionados con IBM MQ classes for JMS/Jakarta Messaging

Se proporcionan varios scripts para ayudar con las tareas comunes que se deben realizar al utilizar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

En la Tabla 14 en la página 128 se incluye la lista de todos los scripts y sus utilizaciones. Los scripts se encuentran en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Tabla 14. Scripts proporcionados con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging

Programa de utilidad	**Utilización**
Cleanup [1]	Este script se conserva para mantener la compatibilidad con releases anteriores, pero no realiza ninguna función. La limpieza manual de la información de suscripción ya no es necesaria.
DefaultConfiguration	Ejecuta la aplicación de configuración predeterminada en plataformas distintas de Windows.
formatLog [1]	Este script se conserva para mantener la compatibilidad con releases anteriores, pero no realiza ninguna función. La salida de la función de registro ahora se genera en texto legible.
IVTRun [1] IVTSetup [1] IVTTidy [1]	Utilizado en la prueba de verificación de la instalación punto a punto, tal como se describe en el apartado "Prueba de verificación de la instalación punto a punto para IBM MQ classes for JMS" en la página 116.
JM 3.0 JMS30Admin [1]	Ejecuta la herramienta de administración de IBM MQ Jakarta Messaging, tal como se describe en Inicio de la herramienta de administración.
JM 3.0 JMS30Admin.config	El archivo de configuración para la herramienta de administración de IBM MQ Jakarta Messaging, tal como se describe en Configuración de la herramienta de administración de JMS.
JMS 2.0 JMSAdmin [1]	Ejecuta la herramienta de administración de IBM MQ JMS, tal como se describe en Inicio de la herramienta de administración.
JMS 2.0 JMSAdmin.config	El archivo de configuración para la herramienta de administración de IBM MQ JMS, tal como se describe en Configuración de la herramienta de administración de JMS.
PSIVTRun [1]	Ejecuta el programa de prueba de verificación de la instalación de publicación/suscripción, tal como se describe en el apartado "Prueba de verificación de la instalación de publicación/suscripción para IBM MQ classes for JMS" en la página 120.
PSReportDump.class	Esta clase se conserva para mantener la compatibilidad con releases anteriores, pero no realiza ninguna función.
V 9.3.0 JM 3.0 V 9.3.0 setjms30env "2" en la página 129	Linux AIX Para Jakarta Messaging 3.0, establece las variables de entorno para ejecutar una aplicación IBM MQ classes for JMS en una máquina virtual Java (JVM) de 32 bits en sistemas AIX and Linux , tal como se describe en "Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97.

Tabla 14. Scripts proporcionados con IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging (continuación)	
Programa de utilidad	**Utilización**
JMS 2.0 setjmsenv "2" en la página 129	Linux AIX Para JMS 2.0, establece las variables de entorno para ejecutar una aplicación IBM MQ classes for JMS en una máquina virtual Java (JVM) de 32 bits en sistemas AIX and Linux , tal como se describe en "Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97.
V 9.3.0 JM 3.0 V 9.3.0 setjms30env64 "2" en la página 129	Linux AIX Para Jakarta Messaging 3.0, establece las variables de entorno para ejecutar una aplicación IBM MQ classes for JMS en una JVM de 64 bits en sistemas AIX and Linux , tal como se describe en "Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97.
JMS 2.0 setjmsenv64 "2" en la página 129	Linux AIX Para JMS 2.0, establece las variables de entorno para ejecutar una aplicación IBM MQ classes for JMS en una JVM de 64 bits en sistemas AIX and Linux , tal como se describe en "Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97.

Nota:

1. En Windows, el nombre de archivo tiene la extensión `.bat`.
2. Estos scripts solo están disponibles en AIX and Linux . En Windows, después de instalar IBM MQ, ejecute el mandato **setmqenv**. Para obtener más información, consulte "Establecimiento de variables de entorno para IBM MQ classes for JMS/Jakarta Messaging" en la página 97.

Soporte de OSGi con IBM MQ classes for JMS

OSGi proporciona una infraestructura que da soporte al despliegue de aplicaciones como paquetes. Los paquetes OSGi se proporcionan como parte de IBM MQ classes for JMS.

IBM MQ classes for JMS incluye los siguientes paquetes OSGi.

com.ibm.msg.client.osgi.jms*version_number*.jar
: La capa común de código en IBM MQ classes for JMS. Para obtener información sobre la arquitectura en capas de las clases de IBM MQ para JMS, consulte Arquitectura de IBM MQ Classes for JMS.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
: Los archivos JAR (archivo Java) de requisito previo de la capa común.

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
: Servicios comunes para aplicaciones de Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_*version_number*.jar
: Mensajes para la capa común.

com.ibm.msg.client.osgi.wmq_*version_number*.jar
: El proveedor de mensajería de IBM MQ en IBM MQ classes for JMS. Para obtener información sobre la arquitectura en capas de IBM MQ classes for JMS, consulte Arquitectura de clases de IBM MQ para JMS.

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
: Los archivos JAR de requisito previo para el proveedor de mensajería de IBM MQ.

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
: Los mensajes del proveedor de mensajería de IBM MQ.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar

V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, este archivo JAR permite a las aplicaciones utilizar IBM MQ classes for JMS y IBM MQ classes for Java, y también incluye el código para manejar mensajes PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar

V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, este archivo JAR proporciona los requisitos previos para `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclient_*version_number*.jar

JMS 2.0 Para JMS 2.0, este archivo JAR permite a las aplicaciones utilizar IBM MQ classes for JMS y IBM MQ classes for Java, y también incluye el código para manejar mensajes PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar

JMS 2.0 Para JMS 2.0, este archivo JAR proporciona los requisitos previos para `com.ibm.mq.osgi.allclient_version_number.jar`.

donde *version_number* es el número de versión de IBM MQ que está instalado.

Los paquetes se instalan en el subdirectorio `java/lib/OSGi` de su instalación de IBM MQ o en la carpeta `java\lib\OSGi` en Windows.

A partir de IBM MQ 8.0, utilice los paquetes `com.ibm.mq.osgi.allclient_8.0.0.0.jar` y `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` para cualquier aplicación nueva. El uso de estos paquetes elimina la restricción de no poder ejecutar IBM MQ classes for JMS y IBM MQ classes for Java en la misma infraestructura de OSGi; no obstante, las demás restricciones continúan aplicándose.

El paquete `com.ibm.mq.osgi.java`*version_number*`.jar`, que también se instala en el subdirectorio `java/lib/OSGi` de la instalación de IBM MQ , o la carpeta `java\lib\OSGi` en Windows, forma parte de IBM MQ classes for Java. Este paquete no debe cargarse en un entorno de ejecución OSGi donde se haya cargado IBM MQ classes for JMS.

Los paquetes OSGi para IBM MQ classes for JMS se han escrito en la especificación OSGi Release 4. No funcionan en un entorno OSGi Release 3.

Debe establecer correctamente la vía de acceso del sistema o de bibliotecas, de forma que el entorno de ejecución OSGi pueda encontrar los archivos DLL o las bibliotecas compartidas necesarias.

Si utiliza los paquetes OSGi para IBM MQ classes for JMS, los temas temporales no funcionan. Además, las clases de salida de canal escritas en Java no están soportadas a causa de un problema inherente a la carga de clases en un entorno de cargador de clases múltiple como OSGi. Un paquete de usuario puede reconocer los paquetes de IBM MQ classes for JMS, pero los paquetes de IBM MQ classes for JMS no reconocen ningún paquete de usuario. Como resultado, el cargador de clases que se utiliza en un paquete de las IBM MQ classes for JMS no puede cargar una clase de salida de canal si está en un paquete de usuario.

Para obtener más información sobre OSGi, consulte el sitio web de OSGi Alliance.

z/OS MQ Adv. VUE Conectividad de cliente JMS/Jakarta Messaging con aplicaciones por lotes que se ejecutan en z/OS

En determinadas condiciones, una aplicación IBM MQ classes for JMS/Jakarta Messaging en z/OS puede conectarse a un gestor de colas en z/OS utilizando una conexión de cliente. El uso de una conexión de cliente puede simplificar las topologías de IBM MQ .

Utilizando una conexión de cliente, si una aplicación IBM MQ classes for JMS/Jakarta Messaging se ejecuta en un entorno de proceso por lotes, y se cumple una de las condiciones siguientes, la aplicación puede conectarse a un gestor de colas z/OS remoto:

- LTS V 9.3.4 El código IBM MQ classes for JMS/Jakarta Messaging está en IBM MQ 9.3.4 o posterior, o Long Term Support con el APAR PH56722 aplicado. El gestor de colas puede estar en cualquier versión soportada.

- El gestor de colas al que se está conectando se está ejecutando con la titularidad de IBM MQ Advanced for z/OS Value Unit Edition y, por lo tanto, tiene el parámetro **ADVCAP** establecido en ENABLED. El gestor de colas puede estar en cualquier versión soportada.

Para obtener más información sobre IBM MQ Advanced for z/OS Value Unit Edition consulte Identificadores de producto deIBM MQ e información de exportación.

Consulte DISPLAY QMGR para obtener más información sobre **ADVCAP** y START QMGR para obtener más información sobre **QMGRPROD**.

Tenga en cuenta que el lote es el único entorno soportado; no hay soporte para JMS/Jakarta Messaging para CICS o JMS/Jakarta Messaging para IMS.

Una aplicación IBM MQ classes for JMS/Jakarta Messaging en z/OS no puede utilizar una conexión en modalidad de cliente para conectarse a un gestor de colas que no se ejecuta en z/OS.

Si una aplicación IBM MQ classes for JMS/Jakarta Messaging en z/OS intenta conectarse utilizando la modalidad de cliente, y no se le permite hacerlo, se emite el mensaje de excepción JMSFMQ0005 .

Soporte de Advanced Message Security (AMS)

Las aplicaciones cliente IBM MQ classes for JMS/Jakarta Messaging pueden utilizar AMS al conectarse a gestores de colas z/OS remotos, sujetos a las condiciones descritas anteriormente en este tema.

Para utilizar AMS de este modo, las aplicaciones cliente deben utilizar un tipo de almacén de claves de `jceracfks` en `keystore.conf`, donde:

- El prefijo de nombre de propiedad es `jceracfks`, en el que no se distingue entre mayúsculas y minúsculas.
- El almacén de claves es un conjunto de claves RACF.
- Las contraseñas no son necesarias y se ignoran. Esto se debe a que los conjuntos de claves RACF no utilizan contraseñas.
- Si se especifica el proveedor, este tiene que ser `IBMJCE`.

Cuando se utiliza `jceracfks` con AMS, el almacén de claves debe tener el formato: `safkeyring://` *`user`*`/`*`keyring`*, donde:

- `safkeyring` es un literal en el que no se distingue entre mayúsculas y minúsculas.
- *`user`* es el ID de usuario de RACF que es propietario del conjunto de claves
- *`keyring`* es el nombre del conjunto de claves RACF y el nombre del conjunto de claves distingue entre mayúsculas y minúsculas

En el ejemplo siguiente se utiliza el conjunto de claves AMS estándar para el usuario `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Conceptos relacionados

"Conectividad de cliente Java con aplicaciones por lotes que se ejecutan en z/OS" en la página 381
En determinadas condiciones, una aplicación IBM MQ classes for Java en z/OS puede conectarse a un gestor de colas en z/OS utilizando una conexión de cliente. El uso de una conexión de cliente puede simplificar las topologías de IBM MQ .

Obtención de IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging por separado

Las bibliotecas y los programas de utilidad necesarios para ejecutar aplicaciones utilizando IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging están disponibles en un archivo JAR autoextraíble que se descarga de Fix Central. Esto se hace si desea obtener sólo estos archivos, por ejemplo, para el despliegue en una herramienta de gestión de software o para utilizarlos con aplicaciones cliente autónomas.

Antes de empezar

Antes de iniciar esta tarea, asegúrese de que tiene instalado un Java runtime environment (JRE) en la máquina y que el JRE se ha añadido a la vía de acceso del sistema.

El instalador de Java que se utiliza en este proceso de instalación no requiere que se ejecute como usuario root ni ningún otro específico. El único requisito es que el usuario con el que se ejecuta tenga acceso de escritura al directorio en el que desea que vayan los archivos.

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

Acerca de esta tarea

Un archivo JAR autoextraíble se utiliza para minimizar el tamaño de la descarga y el tiempo que se tarda en realizar la extracción. El contenido exacto de este archivo JAR y los subdirectorios en los que extrae los archivos dependen de la versión de IBM MQ.

Cuando ejecuta el archivo JAR autoextraíble, muestra el acuerdo de licencia de IBM MQ , que debe aceptarse. También le permite cambiar el directorio padre para la extracción.

Para IBM MQ 9.3, el archivo JAR autoextraíble extrae los archivos en la siguiente estructura de directorios:

wmq/JavaEE
Los archivos EAR y RAR del adaptador de recursos de IBM MQ .

JMS 2.0 Los archivos siguientes se pueden utilizar con objetos JMS 2.0 y JMS 1.1 :

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 También hay un conjunto equivalente para utilizar con objetos Jakarta Messaging 3.0 :

- `wmq.jakarta.jmsra.ivt.ear`. Contiene archivos de prueba de verificación de instalación.
- `wmq.jakarta.jmsra.rar`. Contiene archivos de adaptador de recursos.

wmq/JavaSE

wmq/JavaSE/bin
Las herramientas **JMSAdmin** y **JMS30Admin** . Se utiliza para definir entidades JNDI que representan objetos JMS o Jakarta Messaging .

JMS 2.0 Los archivos siguientes se pueden utilizar con objetos JMS 2.0 y JMS 1.1 :

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 También hay un conjunto equivalente para utilizar con objetos Jakarta Messaging 3.0 :

- `JMS30Admin.bat`. Un archivo que se utiliza para iniciar la herramienta en Windows.
- `JMS30Admin`. Un script que se utiliza para iniciar la herramienta en plataformas Linux y UNIX .
- `JMS30Admin.config`. Un archivo de configuración de ejemplo para la herramienta.

Nota:

- Antes de añadir la herramienta **JMSAdmin** al archivo JAR autoextraíble, los archivos de este directorio estaban en el directorio padre `wmq/JavaSE`.

- Un cliente que se instala utilizando el archivo JAR autoextraíble puede utilizar la herramienta **JMSAdmin** o **JMS30Admin** para crear objetos administrados de mensajería de Java dentro de un contexto de sistema de archivos (archivo `.bindings`). El cliente también puede buscar y utilizar estos objetos administrados.
- JMS 2.0 La herramienta **JMSAdmin** para utilizarla con objetos JMS 2.0 y JMS 1.1 se ha añadido al archivo JAR autoextraíble en IBM MQ 9.2.0 Fix Pack 2 y IBM MQ 9.2.2.
- JM 3.0 La herramienta **JMS30Admin** para utilizarla con objetos Jakarta Messaging 3.0 se ha añadido al archivo JAR autoextraíble en IBM MQ 9.3.0.

wmq/JavaSE/lib

Advanced Message Security utiliza los siguientes paquetes Bouncy Castle de código abierto para dar soporte a Cryptographic Message Syntax (CMS). Consulte Soporte para JRE que no son deIBM con AMS.

V 9.3.5 Para Continuous Delivery desde IBM MQ 9.3.5:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS Para Long Term Support y Continuous Delivery antes de IBM MQ 9.3.5:

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

Cada uno de los archivos siguientes contiene las clases para su nivel JMS o Jakarta Messaging específico:

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 y JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Otros archivos JAR de requisito previo:

- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`. Se utiliza para controlar dinámicamente el rastreo para aplicaciones IBM MQ classes for JMS .
- `fscontext.jar`. Necesario si la aplicación realiza búsquedas JNDI utilizando un contexto de sistema de archivos.
- V 9.3.3 `jackson-annotations.jar`, `jackson-core.jar`, `jackson-databind.jar`: Contiene clases que se utilizan para realizar correlaciones de CipherSuite y CipherSpec al crear conexiones TLS seguras con un gestor de colas.
- JM 3.0 `jakarta.jms-api.jar`. Contiene la interfaz Jakarta Messaging 3.0 y las definiciones de excepción.
- JMS 2.0 `jms.jar`. Contiene la interfaz JMS 2.0 y las definiciones de excepción.
- `org.json.jar`. Contiene clases que permiten a IBM MQ classes for JMS interpretar archivos CCDT de formato JSON.
- `providerutil.jar`. Necesario si la aplicación realiza búsquedas JNDI utilizando un contexto de sistema de archivos.

Nota: Stabilized `com.ibm.mq.allclient.jar` y `com.ibm.mq.jakarta.client.jar` contienen una copia de IBM MQ classes for Java. Sin embargo, en IBM MQ 9.0, estas clases se declaran como estabilizadas funcionalmente en el nivel suministrado en IBM MQ 8.0. Consulte En desuso, estabilizaciones y eliminaciones en IBM MQ 9.0.

wmq/OSGi

Los paquetes de cliente OSGi de IBM MQ :

- JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar
- JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar
- JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar
- JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

donde *V.R.M.F* es el número de Versión, Release, Modificación y Fixpack.

Procedimiento

1. Descargue el archivo JAR de cliente IBM MQ Java / JMS desde Fix Central.
   a) Pulse este enlace: Cliente IBM MQ Java / JMS.
   b) Busque el cliente para su versión de IBM MQ en la lista visualizada de arreglos disponibles.
      Por ejemplo:

      ```
      release level: 9.3.0.0-IBM-MQ-Install-Java-All
      Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
      ```

      A continuación, pulse el nombre del archivo cliente y siga el proceso de descarga.
2. Inicie la extracción desde el directorio en el que ha descargado el archivo.
   Para iniciar la extracción, especifique un mandato con el formato siguiente:

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

   donde *V.R.M.F* es el número de versión del producto, por ejemplo `9.3.0.0`, y *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` es el nombre del archivo que se ha descargado de Fix Central.
   Por ejemplo, para extraer el cliente JMS para el release IBM MQ 9.3.0 , utilizaría el mandato siguiente:

   ```
   java -jar 9.3.0.0-IBM-MQ-Install-Java-All.jar
   ```

   Nota: Para realizar esta instalación, debe tener instalado un JRE en su equipo y añadido a la vía de acceso del sistema.

   Cuando se especifica el mandato, se muestra la siguiente información:

   ```
   Para poder utilizar, extraer o instalar IBM MQ V9.3, debe aceptar
   los términos de 1. IBM Acuerdo Internacional de Licencia para la Evaluación de
   Programas 2. IBM y
   información adicional de licencia. Lea detenidamente los siguientes acuerdos de licencia.

   El acuerdo de licencia se puede ver por separado utilizando la
   opción --viewLicenseAgreement.

   Pulse Intro para mostrar ahora los términos de la licencia o 'x' para omitir.
   ```
3. Revise y acepte los términos de la licencia:
   a) Para ver la licencia, pulse Intro.
      De forma alternativa, pulse x para omitir la visualización de la licencia.
      Después de que se visualice la licencia, o inmediatamente si pulsa x, se visualiza el siguiente mensaje:

      ```
      La información de licencia adicional se puede ver por separado utilizando la
      opción --viewLicenseInfo.

      Pulse Intro para mostrar ahora la información de la licencia adicional o 'x' para omitir.
      ```
   b) Para ver los términos de licencia adicionales, pulse Intro.
      De forma alternativa, pulse x para omitir la visualización de los términos de licencia adicionales.

Después de que se visualicen los términos de licencia adicionales, o inmediatamente si pulsa x, se muestra el siguiente mensaje:

```
Al elegir la opción de "Acepto" a continuación, acepta los términos del
acuerdo de licencia y los términos que no son de IBM, si procede. Si no
está de acuerdo, seleccione "No acepto".

Seleccione [1] Acepto, o [2] No acepto:
```

c) Para aceptar el acuerdo de licencia y continuar con la selección del directorio de instalación, seleccione 1.

De forma alternativa, seleccione 2 para finalizar la instalación inmediatamente.

Si selecciona 1, se visualiza un mensaje similar al siguiente:

```
Especifique el directorio para los archivos de producto o déjelo en blanco para aceptar los
valores predeterminados.
El directorio de destino predeterminado es H: \downloads

¿Directorio de destino para los archivos del producto?
```

4. Especifique el directorio padre para la extracción.

La ubicación predeterminada es el directorio actual.

- Si desea extraer los archivos del producto en la ubicación predeterminada, pulse Intro sin especificar un valor.
- Si desea extraer los archivos del producto en una ubicación diferente, especifique el nombre del directorio en el que desea extraer los archivos y, a continuación, pulse Intro para iniciar la extracción.

El nombre de directorio que especifique no debe existir ya; de lo contrario, cuando inicie la extracción, se informará de un error y no se instalará ningún archivo.

Siempre que no exista todavía, se crea el directorio especificado y los archivos de programa se extraen en este directorio. Durante la instalación, se crea un nuevo directorio con el nombre `wmq` dentro del directorio padre que ha especificado.

Se crean tres subdirectorios, `JavaEE`, `JavaSE`y `OSGi`, en el directorio `wmq` con el contenido siguiente:

JavaEE

JM 3.0 `wmq.jakarta.jmsra.ivt.ear`

JM 3.0 `wmq.jakarta.jmsra.rar`

JMS 2.0 `wmq.jmsra.ivt.ear`

JMS 2.0 `wmq.jmsra.rar`

JavaSE

Este directorio contiene los siguientes subdirectorios y archivos:

JavaSE/lib

V 9.3.5 `bcpkix-jdk18on.jar`

LTS `bcpkix-jdk15on.jar`

V 9.3.5 `bcprov-jdk18on.jar`

LTS `bcprov-jdk15on.jar`

V 9.3.5 `bcutil-jdk18on.jar`

LTS `bcutil-jdk15on.jar`

JMS 2.0 `com.ibm.mq.allclient.jar`

JM 3.0 `com.ibm.mq.jakarta.client.jar`

V 9.3.3 Removed `com.ibm.mq.traceControl.jar`

fscontext.jar

V 9.3.3 jackson-annotations.jar

V 9.3.3 jackson-core.jar

V 9.3.3 jackson-databind.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

Cuando se completa la extracción, se visualiza un mensaje de confirmación tal como se muestra en el ejemplo siguiente:

```
Extrayendo archivos en H: \downloads\wmq
Se han extraído satisfactoriamente todos los archivos del producto.
```

Lista de elementos permitidos en IBM MQ classes for JMS/Jakarta Messaging

El mecanismo de serialización y deserialización de objetos de Java se ha identificado como un riesgo de seguridad potencial. La lista de elementos permitidos en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging proporciona cierta protección contra algunos riesgos de serialización.

Acerca de esta tarea

El mecanismo de deserialización y serialización de objetos Java se ha identificado como un posible riesgo de seguridad porque la deserialización crea instancias arbitrarias de objetos Java, e n los que existe la posibilidad de que se envían datos de forma maliciosa para provocar diversos problemas. Una aplicación notable de serialización se encuentra en Jakarta Messaging 3.0 y Java Message Service 2.0 ObjectMessages que utilizan la serialización para encapsular y transferir objetos arbitrarios.

La lista de elementos permitidos de serialización es una posible mitigación frente a algunos de los riesgos que plantea la serialización. Al especificar explícitamente qué clases se pueden encapsular y extraer de ObjectMessages, la lista de elementos permitidos proporciona cierta protección frente a algunos riesgos de serialización.

Conceptos relacionados

“Ejecución de aplicaciones de IBM MQ classes for JMS en el Java security manager” en la página 111
IBM MQ classes for JMS puede ejecutarse con el Java security manager habilitado. Para ejecutar correctamente aplicaciones con el Java security manager habilitado, debe configurar la Java Virtual Machine (JVM) con un archivo de configuración de políticas adecuado.

Conceptos de la lista de elementos permitidos

En IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, hay soporte para la lista de elementos permitidos de clases en la implementación de la interfaz JMS ObjectMessage . Esto proporciona una mitigación potencial frente a algunos de los riesgos de seguridad que potencialmente están relacionados con el mecanismo de serialización y deserialización de objetos de Java .

Lista de elementos permitidos en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging

Importante:

Siempre que ha sido posible, el término *lista de elementos permitidos* ha sustituido el término *lista blanca*. Para IBM MQ 9.0 y releases posteriores, esto incluye algunos nombres de propiedad del sistema Java mencionados en este tema. No tiene que cambiar ninguna configuración existente. Los nombres de propiedad del sistema anteriores también siguen funcionando.

IBM MQ classes for JMS (JMS 2.0) V 9.3.0 V 9.3.0 y IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) dan soporte a la lista de elementos permitidos de clases en la implementación de la interfaz JMS ObjectMessage .

- JMS 2.0 Para IBM MQ classes for JMS, los nombres de propiedad relevantes son **`com.ibm.mq.jms.allowlist.*`**.
- JM 3.0 Para IBM MQ classes for Jakarta Messaging, los nombres de propiedad relevantes son **`com.ibm.mq.jakarta.jms.allowlist.*`**

La lista de elementos permitidos define qué clases Java se pueden serializar con ObjectMessage.setObject() y deserializar con ObjectMessage.getObject().

- JMS 2.0 Los intentos de serializar o deserializar una instancia de una clase no incluida en la lista de elementos permitidos con ObjectMessage hacen que se genere una excepción javax.jms.MessageFormatException , con una excepción java.io.InvalidClassException como causa.
- JM 3.0 Los intentos de serializar o deserializar una instancia de una clase no incluida en la lista de elementos permitidos con ObjectMessage hacen que se emita una excepción jakarta.jms.MessageFormatException , con una excepción java.io.InvalidClassException como causa.

Producción de la lista de elementos permitidos

Importante: IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging no se pueden distribuir con una lista de elementos permitidos. La elección de las clases que se van a transferir utilizando ObjectMessages es una opción de diseño de aplicaciones y IBM MQ no puede anticiparla.

Por esta razón, el mecanismo de lista de elementos permitidos permite dos modalidades de operación:

DISCOVERY
: En este modo, el mecanismo produce un listado de nombres de clase totalmente cualificados, que informa de todas las clases que se han observado para ser serializadas o deserializadas en ObjectMessages.

ENFORCEMENT
: En esta modalidad, el mecanismo aplica la lista de elementos permitidos, rechazando los intentos de serializar o deserializar las clases que no están en la lista de elementos permitidos.

Si desea utilizar este mecanismo, debe ejecutar inicialmente en modalidad DISCOVERY para recopilar la lista de clases serializadas y deserializadas actualmente, revisar la lista y utilizarla como base para la lista de elementos permitidos. Dicha lista podría usarse incluso sin modificaciones, pero antes hay que revisarla para decidirlo.

Control del mecanismo de lista de elementos permitidos

Hay tres propiedades del sistema disponibles para controlar el mecanismo de lista de elementos permitidos:

com.ibm.mq.jms.allowlist (JMS 2.0) y com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
: Esta propiedad se puede especificar de una de las formas siguientes:

  - El nombre de vía de acceso del archivo que contiene la lista de elementos permitidos, en formato de URI de archivo (es decir, empezando por `file:`). En la modalidad DISCOVERY, el mecanismo

de lista de elementos permitidos escribe en este archivo. El archivo no puede existir. Si existiera, el mecanismo generaría una excepción en lugar de sobrescribirlo. En la modalidad ENFORCEMENT, el mecanismo de lista de elementos permitidos lee este archivo.

- Una coma separada de nombres de clase completos que constituyen la lista de elementos permitidos.

Si esta propiedad no está establecida, el mecanismo de lista de elementos permitidos está inactivo.

Si está utilizando Java security manager, debe asegurarse de que los archivos JAR IBM MQ classes for JMS tienen acceso de lectura y escritura a este archivo.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) y com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- Si esta propiedad no se establece o se establece en false, el mecanismo de lista de elementos permitidos se ejecuta en modalidad ENFORCEMENT.
- Si esta propiedad se establece en true y la lista de elementos permitidos se ha especificado como un URI de archivo, el mecanismo de lista de elementos permitidos se ejecuta en modalidad DISCOVERY.
- Si esta propiedad se establece en true y la lista de elementos permitidos se ha especificado como una lista de nombres de clase, el mecanismo de lista de elementos permitidos genera una excepción adecuada.
- Si esta propiedad se establece en true y la lista de elementos permitidos no se ha especificado utilizando la propiedad com.ibm.mq.jms.allowlist o com.ibm.mq.jakarta.jms.allowlist, el mecanismo allowlist está inactivo.
- Si esta propiedad se establece en true y el archivo allowlist ya existe, el mecanismo allowlist genera una excepción java.io.InvalidClassException y las entradas no se añaden al archivo.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) y com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)

Esta propiedad de cadena se puede especificar de tres maneras:

- Si esta propiedad se establece en SERIALIZE, la modalidad ENFORCEMENT sólo realiza la validación de la lista de elementos permitidos en el método ObjectMessage.setObject().
- Si esta propiedad se establece en DESERIALIZE, la modalidad ENFORCEMENT sólo realiza la validación de la lista de elementos permitidos en el método ObjectMessage.getObject().
- Si esta propiedad no se establece, o se establece en cualquier otro valor, la modalidad ENFORCEMENT realiza la validación de lista de elementos permitidos en los métodos ObjectMessage.getObject() y ObjectMessage.setObject().

Formato del archivo de lista de elementos permitidos

Estas son las características principales del formato del archivo de lista de elementos permitidos:

- El archivo de lista de elementos permitidos está en la codificación de archivos de plataforma predeterminada con finales de línea adecuados para la plataforma.

  Nota: Si se está utilizando un archivo de lista de elementos permitidos, dicho archivo siempre se graba y se lee utilizando la codificación de archivo predeterminada para la JVM.

  Esto es correcto si el archivo de lista de elementos permitidos se genera de alguna de las maneras siguientes:

  - z/OS Generado por una aplicación autónoma que se ejecuta en z/OS y es utilizado por otras aplicaciones autónomas que también se ejecutan en z/OS.
  - Generado por una aplicación que se ejecuta dentro de WebSphere Application Server en cualquier plataforma, y es utilizado por otra instancia de WebSphere Application Server.

  - Generado por una aplicación autónoma que se ejecuta en IBM MQ for Multiplatforms, y es utilizado por otras aplicaciones autónomas que se ejecutan en IBM MQ for Multiplatforms, o por aplicaciones que se ejecutan dentro de WebSphere Application Server en cualquier plataforma.

  Sin embargo, como WebSphere Application Server utiliza ASCII, y una JVM autónoma utiliza EBCDIC, habrá problemas de codificación de archivos si el archivo de lista de elementos permitidos se genera de una de las formas siguientes:

  - Se genera en z/OS y, después, es utilizado por aplicaciones autónomas que se ejecutan en una plataforma distinta a z/OS o por WebSphere Application Server.
  - Generado por WebSphere Application Server o una aplicación autónoma que se ejecuta en una plataforma distinta a z/OS y, después, es utilizado por una aplicación autónoma en z/OS.
- Cada línea no vacía contiene un nombre de clase totalmente cualificado. Las líneas vacías se ignoran.
- Se pueden incluir comentarios (cualquier cosa que siga a un carácter '#', al final de la línea, se ignora).
- Hay un mecanismo de uso de comodines muy básico:
  - '*' puede ser el **último** elemento de un nombre de clase.
  - '*' coincide con un **único** elemento del nombre de clase, es decir, la clase, pero sin paquete.

  Así, `com.ibm.mq.*` coincidiría con `com.ibm.mq.MQMessage`, pero no con `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  El uso de comodines no funciona con clases del paquete predeterminado, es decir, clases sin un nombre de paquete explícito, de forma que se rechazaría el nombre de clase "*".
- Los archivos de lista de elementos permitidos con formato incorrecto, por ejemplo, los archivos que contienen una entrada como `com.ibm.mq.*.Message`, donde el comodín no es el último elemento, hacen que se emita una excepción java.lang.IllegalArgumentException .
- Un archivo de lista de elementos permitidos vacío tiene el efecto de inhabilitar totalmente el uso de ObjectMessage.

Formato de la lista de elementos permitidos como una lista separada por comas

El mismo mecanismo de comodín está disponible para una lista de elementos permitidos como una lista separada por comas.

- '*' puede ser expandido por el sistema operativo si se especifica en una línea de comandos o en un script de shell o en un archivo de proceso por lotes, por lo que podría requerir un tratamiento especial.
- El carácter de comentario '#' solo es aplicable cuando se especifica un archivo. Si la lista de elementos permitidos se especifica como una lista separada por comas de nombres de clase, suponiendo que el sistema operativo o shell no la procesa, ya que es el carácter de comentario predeterminado en muchos shells AIX and Linux , se trata como un carácter normal.

¿Cuándo se produce la lista de elementos permitidos?

La lista de elementos permitidos se inicia cuando la aplicación ejecuta por primera vez un método ObjectMessage setMessage() o getMessage().

Se evalúan las propiedades del sistema, se abre el archivo de lista de elementos permitidos y, en modalidad ENFORCEMENT, se carga la lista de clases de la lista de elementos permitidos cuando se inicializa el mecanismo. En este punto, se escribe una entrada en el archivo de registro IBM MQ JMS para la aplicación.

Cuando el mecanismo se ha inicializado, sus parámetros no se pueden cambiar. El tiempo de inicialización no se puede predecir fácilmente, ya que depende del comportamiento de la aplicación. Por lo tanto, los valores de propiedad del sistema y el contenido del archivo de lista de elementos permitidos deben considerarse arreglados desde el momento en que se inicia la aplicación. No cambie las propiedades ni el contenido del archivo de lista de elementos permitidos mientras se ejecuta la aplicación, ya que los resultados no están garantizados.

Puntos por tener en cuenta

El mejor enfoque para mitigar los riesgos intrínsecos al mecanismo de serialización de Java sería explorar enfoques alternativos a la transferencia de datos como, por ejemplo, utilizando JSON en lugar de ObjectMessage. El uso de mecanismos de Advanced Message Security (AMS) puede añadir seguridad adicional al garantizar que los mensajes proceden de orígenes de confianza.

Si utiliza el mecanismo Java security manager con la aplicación, debe otorgar los permisos siguientes:

- FilePermission en cualquier archivo de lista de elementos permitidos que utilice, con permiso de lectura para la modalidad ENFORCEMENT, permiso de escritura para la modalidad DISCOVER.
- JMS 2.0 PropertyPermission (lectura) en las propiedades **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover**y **com.ibm.mq.jms.allowlist.mode** .
- JM 3.0 PropertyPermission (lectura) en las propiedades **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover**y **com.ibm.mq.jakarta.jms.allowlist.mode** .

Más información

Consulte “Configuración y utilización de una lista de elementos permitidos JMS o Jakarta Messaging” en la página 140 y “Lista de elementos permitidos en WebSphere Application Server” en la página 142 para obtener más información sobre las listas de elementos permitidos.

Conceptos relacionados

“Ejecución de aplicaciones de IBM MQ classes for JMS en el Java security manager” en la página 111
IBM MQ classes for JMS puede ejecutarse con el Java security manager habilitado. Para ejecutar correctamente aplicaciones con el Java security manager habilitado, debe configurar la Java Virtual Machine (JVM) con un archivo de configuración de políticas adecuado.

Configuración y utilización de una lista de elementos permitidos JMS o Jakarta Messaging

Esta información le indica cómo funciona una lista de elementos permitidos y cómo se configura una utilizando la funcionalidad contenida en IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging para generar un archivo de lista de elementos permitidos, que contiene una lista de los tipos de ObjectMessages que una aplicación puede procesar.

Antes de empezar

Importante:

Siempre que ha sido posible, el término *lista de elementos permitidos* ha sustituido el término *lista blanca*. Para IBM MQ 9.0 y releases posteriores, esto incluye algunos nombres de propiedad del sistema Java mencionados en este tema. No tiene que cambiar ninguna configuración existente. Los nombres de propiedad del sistema anteriores también siguen funcionando.

Antes de iniciar esta tarea, asegúrese de que ha leído y comprendido “Conceptos de la lista de elementos permitidos” en la página 136

Acerca de esta tarea

Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

Cuando haya habilitado la funcionalidad de lista de elementos permitidos, IBM MQ classes for JMS utilizará dicha funcionalidad de las siguientes maneras:

- Cuando una aplicación desea enviar un ObjectMessage, puede crearlo de una de dos formas, llamando al:

– método Session.createObjectMessage(Serializable), que se pasa en el objeto que se va a incluir en el mensaje.
– método Session.createObjectMessage(), para crear un ObjectMessage vacío y, después, llamando a ObjectMessage.setObject(Serializable) para almacenar el objeto que se va a pasar dentro de ObjectMessage.

Cuando se llama a los métodos Session.createObjectMessage(Serializable) o ObjectMessage.setObject(Serializable), las clases para JMS comprueban si el objeto pasado es de un tipo que se menciona en la lista de elementos permitidos.

Si de un tipo mencionado, el objeto se serializa y se almacena en ObjectMessage. Sin embargo, si el objeto es de un tipo que no está en la lista de elementos permitidos, IBM MQ classes for JMS genera una excepción JMSException que contiene el mensaje:

```
JMSCC0052: se ha producido una excepción al serializar el objeto:
'java.io.InvalidClassException: <object class>; La clase no se puede serializar
o deserializado, ya que no se ha incluido en la lista de elementos permitidos '< allowlist>'.
```

Volver a la aplicación.

Importante: Si la excepción se lanza desde el método Session.createObjectMessage(Serializable), el ObjectMessage no se creará. De forma similar, si se lanza JMSException desde el método ObjectMessage.setObject(Serializable), el objeto no se añadirá al ObjectMessage.

- Si una aplicación recibe un ObjectMessage, llama al método ObjectMessage.getObject() para obtener el objeto incluido en el mismo. Cuando se llama a este método, IBM MQ classes for JMS comprueba el tipo de objeto contenido en el ObjectMessage, para ver si ese objeto es de un tipo especificado en la lista de elementos permitidos.

  En caso afirmativo, el objeto se deserializa y se devuelve a la aplicación. Sin embargo, si el objeto es de un tipo que no está en la lista de elementos permitidos, IBM MQ classes for JMS genera una excepción JMSException que contiene el mensaje:

  ```
  JMSCC0053: se ha producido una excepción al deserializar un mensaje:
  'java.io.InvalidClassException: <object class>; la clase no se puede
  serializar ni deserializar ya que no se ha incluido en la
  allowlist '< listapermio>'. '.
  ```

  Volver a la aplicación.

Por ejemplo, suponga que la aplicación contiene el código siguiente para enviar un ObjectMessage que contiene un objeto de tipo java.net.URI:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Puesto que la lista de elementos permitidos no está habilitada, la aplicación puede colocar correctamente el mensaje en el destino necesario.

Si crea un archivo denominado `C:\allowlist.txt` que contiene una sola entrada, `java.net.URL`, y vuelve a iniciar la aplicación con la propiedad del sistema Java establecida:

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

la funcionalidad de la lista de elementos permitidos está habilitada. La aplicación todavía puede crear y enviar el ObjectMessage que contiene un objeto de tipo java.net.URI , ya que dicho tipo se especifica en la lista de elementos permitidos.

Sin embargo, si cambia el archivo `allowlist.txt` para que el archivo contenga la entrada única `java.util.Calendar`, ya que la funcionalidad de la lista de elementos permitidos sigue habilitada, cuando la aplicación llama:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS compruebe la lista de elementos permitidos y compruebe que no contiene una entrada para java.net.URI.

Como resultado, se lanza una excepción JMSException que contiene el mensaje JMSCC0052.

De forma similar, suponga que tiene otra aplicación que recibe ObjectMessages que utiliza este código:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Si la lista de elementos permitidos no está habilitada, la aplicación puede recibir ObjectMessages que contengan un objeto de cualquier tipo. A continuación, la aplicación comprueba si el objeto es del tipo java.net.URL antes de realizar el proceso apropiado.

Si ahora inicia la aplicación con la propiedad del sistema Java:

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

, la funcionalidad de lista de elementos permitidos está activada. Cuando la aplicación llama a:

```
Object messageBody = objectMessage.getObject();
```

el método ObjectMessage.getObject() solo devuelve objetos del tipo java.net.URL.

Si el objeto incluido en el ObjectMessage no es de este tipo, el método ObjectMessage.getObject() lanza una excepción JMSException que contiene el mensaje JMSCC0053. A continuación, la aplicación decide qué hacer con el mensaje; por ejemplo, el mensaje se ha podido trasladar a la cola de mensajes no entregados para dicho gestor de colas.

La aplicación solo realiza un retorno de forma normal si el objeto dentro del ObjectMessage no es del tipo java.net.URL.

Procedimiento

1. Ejecute la aplicación que procesa ObjectMessages, con las siguientes propiedades del sistema Java especificadas:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Cuando se ejecuta la aplicación, IBM MQ classes for JMS crean un archivo que contiene los tipos de objetos que ha procesado la aplicación.
2. Después de que la aplicación haya procesado un ejemplo representativo de ObjectMessages durante un periodo de tiempo, deténgala.

   El archivo de lista de elementos permitidos ahora contiene una lista de todos los tipos de objetos contenidos en los ObjectMessages que la aplicación ha procesado mientras se estaba ejecutando.

   Si ha ejecutado la aplicación durante un periodo de tiempo suficiente, esta lista incluye todos los tipos posibles de objetos en ObjectMessages que probablemente va a manejar la aplicación.
3. Reinicie la aplicación con la propiedad de sistema siguiente establecida:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Esto habilita la lista de elementos permitidos y, si IBM MQ classes for JMS detecta un ObjectMessage de un tipo que no está en la lista de elementos permitidos, se genera una excepción JMSException que contiene el mensaje JMSCC0052 o JMSCC0053 .

Lista de elementos permitidos en WebSphere Application Server

Cómo utilizar la lista de elementos permitidos de IBM MQ classes for JMS en WebSphere Application Server.

Importante:

Siempre que ha sido posible, el término *lista de elementos permitidos* ha sustituido el término *lista blanca*. Para IBM MQ 9.0 y releases posteriores, esto incluye algunos nombres de propiedad del sistema Java mencionados en este tema. No tiene que cambiar ninguna configuración existente. Los nombres de propiedad del sistema anteriores también siguen funcionando.

Debe asegurarse de que la instalación de WebSphere Application Server incluye una versión del adaptador de recursos de IBM MQ que da soporte a la lista de elementos permitidos.

Consulte "Utilización de IBM MQ y WebSphere Application Server juntos" en la página 515 para obtener más información sobre cómo utilizar los dos productos.

A partir de IBM MQ 9.0.0 Fix Pack 1 hacia adelante, se incluye la funcionalidad apropiada.

Una vez que el servidor de aplicaciones se haya actualizado, puede utilizar las propiedades del sistema Java:

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

descrito en "Configuración y utilización de una lista de elementos permitidos JMS o Jakarta Messaging" en la página 140.

Nota: Necesita establecer las propiedades del sistema Java como argumentos de JVM genéricos, en Java Virtual Machine se utiliza para ejecutar el servidor de aplicaciones y el servidor de aplicaciones se reinicia para que los cambios entren en vigor.

Consulte la sección en *Argumentos de JVM genéricos* en Valores de la máquina virtual Java para obtener más información.

Para establecer las propiedades, vaya a la ventana Java Virtual Machine en *Definiciones de proceso* y especifique el argumento apropiado.

El valor siguiente:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

hace que el servidor de aplicaciones utilice la lista de elementos permitidos *youruserId_MyObject*. El servidor de aplicaciones solo procesa los objetos de ese tipo.

Los valores siguientes:

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

configurar el servidor de aplicaciones para que utilice la modalidad *Descubrir* y registrar los detalles de JMS ObjectMessages, que procesa el servidor de aplicaciones, en el archivo `C:\allowlist.txt`

El valor siguiente:

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

hace que el servidor de aplicaciones cargue el archivo `C:/allowlist.txt`y utilice la información de dicho archivo para determinar la lista de elementos permitidos.

Conceptos relacionados

"Ejecución de aplicaciones de IBM MQ classes for JMS en el Java security manager" en la página 111
IBM MQ classes for JMS puede ejecutarse con el Java security manager habilitado. Para ejecutar correctamente aplicaciones con el Java security manager habilitado, debe configurar la Java Virtual Machine (JVM) con un archivo de configuración de políticas adecuado.

Conversiones de cadenas de caracteres en IBM MQ classes for JMS

Los IBM MQ classes for JMS utilizan CharsetEncoders y CharsetDecoders directamente para la conversión de series de caracteres. El comportamiento predeterminado para la conversión de series de caracteres

se puede configurar con dos propiedades del sistema. El manejo de mensajes que contienen caracteres no correlacionables se puede configurar mediante propiedades de mensaje para establecer la acción UnmappableCharactery los bytes de sustitución.

Antes de IBM MQ 8.0, las conversiones de series en IBM MQ classes for JMS se realizaban llamando a los métodos `java.nio.charset.Charset.decode(ByteBuffer)` y `Charset.encode(CharBuffer)`.

La utilización de cualquiera de estos métodos da lugar a una sustitución predeterminada (REPLACE) de datos malformados o no traducibles. Este comportamiento puede ocultar errores en las aplicaciones y dar lugar a caracteres inesperados, por ejemplo ?, en los datos traducidos.

A partir de IBM MQ 8.0, para detectar dichos problemas de forma más rápida y eficaz, IBM MQ classes for JMS utilizan CharsetEncoders y CharsetDecoders directamente y configuran explícitamente el manejo de los datos mal formados y no trducibles. El comportamiento predeterminado es REPORT estos problemas emitiendo un `MQException`adecuado.

Configuración

La conversión de UTF-16 (la representación de caracteres utilizada en Java) z un juego de caracteres nativos como, UTF-8, se denomina *codificación*, mientras que la conversión en dirección opuesta se denomina *descodificación*.

La decodificación adopta el comportamiento predeterminado para `CharsetDecoders`, notificando errores emitiendo una excepción.

Se usa un parámetro de configuración para especificar una `java.nio.charset.CodingErrorAction` que controle el manejo de errores en la codificación y en la descodificación. Se usa otro parámetro de configuración para controlar el byte, o los bytes, de sustitución al codificar. Se usará la cadena Java de sustitución predeterminada en las operaciones de decodificación.

UnmappableCharacterValores de bytes de acción y sustitución en IBM MQ classes for JMS

A partir de IBM MQ 8.0, las dos propiedades siguientes están disponibles para establecer la acción UnmappableCharactery los bytes de sustitución. Las definiciones de constante adecuadas se encuentran en `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_UNMAPPABLE_ACTION
Establece o obtiene la `CodingErrorAction` que se aplica cuando no se puede correlacionar un carácter con una operación de codificación o decodificación.

Hay que establecerla a `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` de la manera siguiente:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
Establece u obtiene los bytes de sustitución que se aplican cuando un carácter no se puede correlacionar con una operación de codificación.

La cadena Java de sustitución predeterminada se usa en las operaciones de decodificación.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Las propiedades `JMS_IBM_UNMAPPABLE_ACTION` y `JMS_IBM_UNMAPPABLE_REPLACEMENT` se pueden establecer en destinos o mensajes. Un valor definido en un mensaje sustituye el valor definido en el destino al que se envía dicho mensaje.

Tenga en cuenta que `JMS_IBM_UNMAPPABLE_REPLACEMENT` tiene que definirse como un único byte.

Propiedades del sistema para establecer valores predeterminados del sistema

A partir de IBM MQ 8.0, las dos propiedades del sistema Java siguientes están disponibles para configurar el comportamiento predeterminado con respecto a la conversión de series de caracteres.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Especifica la acción que hay que realizar con los datos no traducibles en codificación y decodificación. El valor puede ser `REPORT`, `REPLACE` o `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Establece u obtiene los bytes de sustitución que se aplican cuando un carácter no se puede correlacionar en una operación de codificación. La serie de sustitución Java predeterminada se utiliza en operaciones de descodificación.

Para evitar confusiones entre las representaciones de caracteres y de bytes nativos de Java, debe especificar `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` como un número decimal que representa el byte de sustitución en el juego de caracteres nativos.

Por ejemplo, el valor decimal de ?, como byte nativo, es 63 si el juego de caracteres nativo está basado en ASCII como, por ejemplo, ISO-8859-1, mientras que es 111 si el juego de caracteres nativo es EBCDIC.

Nota: Tenga en cuenta que si un objeto MQMD o MQMessage tiene establecidos los campos **`unmappableAction`** o **`unMappableReplacement`** , los valores de estos campos tienen prioridad sobre las propiedades del sistema Java . Esto permite que los valores especificados por las propiedades del sistema Java se alteren temporalmente para cada mensaje si es necesario.

Conceptos relacionados

"Conversiones de cadenas de caracteres en IBM MQ classes for Java" en la página 360
Los IBM MQ classes for Java utilizan CharsetEncoders y CharsetDecoders directamente para la conversión de series de caracteres. El comportamiento predeterminado para la conversión de series de caracteres se puede configurar con dos propiedades del sistema. El manejo de mensajes que contienen caracteres no correlacionables se puede configurar a través de `com.ibm.mq.MQMD`.

Escritura de aplicaciones IBM MQ classes for JMS/Jakarta Messaging

Después de una breve introducción al modelo JMS , esta sección proporciona una guía detallada sobre cómo escribir aplicaciones IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging .

Acerca de esta tarea

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

Conceptos relacionados

IBM MQ classes for Jakarta Messaging: una visión general

El modelo JMS y Jakarta Messaging

El modelo JMS y Jakarta Messaging define un conjunto de interfaces que las aplicaciones Java pueden utilizar para realizar operaciones de mensajería. IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son ambos proveedores de mensajería. Definen cómo se relacionan los objetos JMS y Jakarta Messaging con los conceptos de IBM MQ . Las especificaciones JMS y Jakarta Messaging esperan que determinados objetos JMS y Jakarta Messaging sean objetos administrados.

JMS 2.0 IBM MQ 8.0 ha añadido soporte para la versión JMS 2.0 del estándar JMS , que ha introducido una API simplificada, al tiempo que conserva la API clásica, de JMS 1.1.

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones

existentes. No está soportado utilizar tanto la API de JMS 2.0 como la API de Jakarta Messaging 3.0 en la misma aplicación.

Nota: Para Jakarta Messaging 3.0, el control de la especificación JMS se mueve de Oracle a Java Community Process. Sin embargo, Oracle conserva el control del nombre "javax", que se utiliza en otras tecnologías Java que no se han movido al proceso de comunidad de Java . Por lo tanto, aunque Jakarta Messaging 3.0 es funcionalmente equivalente a JMS 2.0 , existen algunas diferencias en la denominación:

- El nombre oficial de la versión 3.0 es Jakarta Messaging en lugar de Java Message Service.
- Los nombres de paquete y constante tienen el prefijo `jakarta` en lugar de `javax`. Por ejemplo, en JMS 2.0 la conexión inicial con un proveedor de mensajería es un objeto `javax.jms.Connection` y en Jakarta Messaging 3.0 es un objeto `jakarta.jms.Connection` .

JMS 2.0 Los paquetes javax.jms definen las interfaces JMS y un proveedor JMS implementa estas interfaces para un producto de mensajería específico. IBM MQ classes for JMS es un proveedor de JMS que implementa interfaces JMS para IBM MQ.

JM 3.0 Los paquetes jakarta.jms definen las interfaces Jakarta Messaging y un proveedor Jakarta Messaging implementa estas interfaces para un producto de mensajería específico. IBM MQ classes for Jakarta Messaging es un proveedor de Jakarta Messaging que implementa interfaces Jakarta Messaging para IBM MQ.

Debido a que JMS y Jakarta Messaging comparten mucho en común, las referencias adicionales a JMS en este tema se pueden tomar como referencias a ambos. Las diferencias se resaltan según sea necesario.

API simplificada

JMS 2.0 ha introducido la API simplificada, al tiempo que retiene las interfaces específicas del dominio e independientes del dominio de JMS 1.1. La API simplificada disminuye el número de objetos necesarios para enviar y recibir mensajes y consta de las interfaces siguientes:

ConnectionFactory
ConnectionFactory es un objeto administrado que utiliza un cliente de JMS para crear una conexión. Esta interfaz también se utiliza en la API clásica.

Contexto de JMS
Este objeto combina los objetos Conexión y Sesión de la API clásica. Se pueden crear objetos JMSContext a partir de otros objetos JMSContext, duplicando la conexión subyacente.

JMSProductor
Se crea un JMSProducer mediante un JMSContext y se utiliza para enviar mensajes a una cola o tema. El objeto JMSProducer genera la creación de los objetos necesarios para enviar el mensaje.

JMSConsumidor
Se crea un JMSConsume mediante un JMSContext y se utiliza para recibir mensajes de un tema o una cola.

La API simplificada tiene varios efectos:

- El objeto JMSContext siempre inicia automáticamente la conexión subyacente.
- Ahora los JMSProducers y los JMSConsumers pueden trabajar directamente con los cuerpos de los mensajes, sin tener que obtener el objeto de mensaje completo, utilizando el método getBody del mensaje.
- Las propiedades de los mensajes se pueden establecer en el objeto JMSProducer mediante el encadenamiento de métodos, antes de enviar un 'body', un contenido de mensajes. El objeto JMSProducer manejará la creación de todos los objetos necesarios para enviar el mensaje. Utilizando JMS 2.0, se pueden establecer las propiedades y un envío de mensajes, como se indica a continuación:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
```

```
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 también introdujo suscripciones compartidas donde los mensajes se pueden compartir entre varios consumidores. Todas las suscripciones de JMS 1.1 se tratan como suscripciones no compartidas.

API clásica

La lista siguiente resume las interfaces principales de JMS de la API clásica:

Destino
Un objeto Destination es la ubicación a la que una aplicación envía mensajes, o es el origen desde el que una aplicación recibe mensajes, o ambas cosas.

ConnectionFactory
Un objeto ConnectionFactory encapsula un conjunto de propiedades de configuración de una conexión. Una aplicación utiliza una fábrica de conexiones para crear una conexión.

Conexión
Un objeto Connection encapsula una conexión activa de una aplicación en un servidor de mensajería. Una aplicación utiliza una conexión para crear sesiones.

Session
Una sesión es un contexto de hebra única para enviar y recibir mensajes. Una aplicación utiliza una sesión para crear mensajes, productores de mensajes y consumidores de mensajes. Una sesión es transaccional o no transaccional.

Mensaje
Un objeto Message encapsula un mensaje que una aplicación envía o recibe.

MessageProducer
Una aplicación utiliza un productor de mensajes para enviar mensajes a un destino.

MessageConsumer
Una aplicación utiliza un consumidor de mensajes para recibir mensajes enviados a un destino.

Figura 9 en la página 147 muestras estos objetos y sus relaciones.

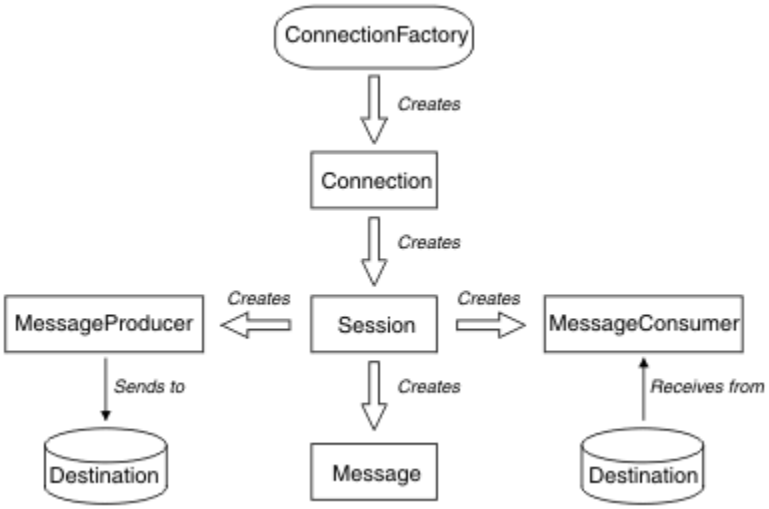


Figura 9. Objetos JMS y sus relaciones

Un objeto Destination, ConnectionFactory o Connection lo pueden utilizar varias hebras al mismo tiempo, pero no pueden utilizar al mismo tiempo un objeto Session, MessageProducer o MessageConsumer. La forma más simple de asegurarse de que un objeto Session, MessageProducer o MessageConsumer no se utiliza simultáneamente es crear un objeto Session separado para cada hebra.

dominios de mensajería

JMS da soporte a dos estilos de mensajería:

- Mensajería punto a punto
- Mensajería de publicación/suscripción

También se hace referencia a estos tipos de mensajería como *dominios de mensajería* y puede combinar ambos tipos de mensajería en una aplicación. En el dominio punto a punto, un destino es una cola y, en el dominio de publicación/suscripción, un destino es un tema.

En las versiones de JMS anteriores a JMS 1.1, la programación para los dominios punto a punto utiliza un conjunto de interfaces y métodos y la programación para los dominios de publicación/suscripción utiliza otro conjunto. Los dos conjuntos son similares, pero independientes. A partir de JMS 1.1, puede utilizar un conjunto común de interfaces y métodos que dan soporte a ambos dominios de mensajería. Las interfaces comunes proporcionan una vista independiente del dominio para cada dominio de mensajería. La Tabla 15 en la página 148 muestra las interfaces independientes del dominio de JMS y sus interfaces específicas del dominio correspondientes.

Tabla 15. Las interfaces independientes del dominio y específicas del dominio de JMS

Interfaces independientes del dominio	**Interfaces específicas del dominio para el dominio punto a punto**	**Interfaces específicas del dominio para el dominio de publicación/suscripción**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Conexión	QueueConnection	TopicConnection
Destino	Cola	Tema
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 da soporte a las interfaces específicas de dominio de JMS 1.1 anteriores y a la API simplificada de JMS 2.0. Por lo tanto, IBM MQ classes for JMS 2.0 se puede utilizar para mantener aplicaciones existentes, incluido el desarrollo de nuevas funciones en aplicaciones existentes.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 da soporte a las versiones de Jakarta Messaging de las mismas interfaces y se recomienda para el desarrollo de nuevas aplicaciones.

En IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, los objetos JMS están relacionados con los conceptos de IBM MQ de las maneras siguientes:

- Un objeto Connection tiene propiedades derivadas de las propiedades de la fábrica de conexiones utilizada para crear la conexión. Estas propiedades controlar cómo se conecta una aplicación a un gestor de colas. Los ejemplos de estas propiedades son el nombre del gestor de colas y, en el caso de una aplicación que se conecta al gestor de colas en modo cliente, el nombre de host o dirección IP del sistema en el que se ejecuta el gestor de colas.
- Un objeto Session encapsula un identificador de conexión IBM MQ, que por lo tanto define el ámbito transaccional de la sesión.
- Un objeto MessageProducer y un objeto MessageConsumer encapsula cada uno un manejador de objetos de IBM MQ.

Cuando se utiliza IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, se aplican todas las reglas normales de IBM MQ . En concreto, tenga en cuenta que una aplicación puede enviar un mensaje a una cola remota pero solo puede recibir un mensaje de una cola propiedad del gestor de colas al que está conectada la aplicación.

La especificación JMS espera que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene los objetos administrados en un repositorio central y una aplicación JMS recupera estos objetos utilizando la interfaz JNDI (Naming and Directory Interface) de Java.

En IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging, la implementación de la interfaz Destination es una superclase abstracta de Queue y Topic, por lo que una instancia de Destination es un objeto Queue o un objeto Topic. La interfaz independiente del dominio trata una cola o un tema como un destino. El dominio de mensajería para un objeto MessageProducer o MessageConsumer queda determinado por si el destino es una cola o un tema.

Por lo tanto, en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging , los objetos de los tipos siguientes pueden ser objetos administrados:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Cola
- Tema
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Conceptos relacionados

Interfaces de lenguaje de IBM MQ Java

"Creación y configuración de fábricas de conexiones y destinos" en la página 210

Una aplicación IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging puede crear fábricas de conexiones y destinos recuperándolos como objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface), utilizando las extensiones de IBM JMS o utilizando las extensiones de IBM MQ JMS . Una aplicación también puede utilizar las extensiones de IBM JMS o las extensiones de IBM MQ JMS para establecer las propiedades de fábricas y destinos de conexiones.

Mensajes de JMS

Los mensajes de JMS se componen de una cabecera, propiedades y un cuerpo. JMS define cinco tipos de cuerpo de mensaje.

Los mensajes de JMS constan de las partes siguientes:

Cabecera
Todos los mensajes dan soporte al mismo conjunto de campos de cabecera. Los campos de cabecera contienen valores que utilizan tanto los clientes como los proveedores para identificar y direccionar mensajes.

Propiedades
Cada mensaje contiene un recurso incorporado para dar soporte a los valores de propiedad que define la aplicación. Las propiedades facilitan un mecanismo eficaz para filtrar los mensajes que define la aplicación.

Cuerpo
JMS define cinco tipos de cuerpo de mensaje que cubren la mayoría de los estilos de mensajería actualmente en uso:

Corriente
Una corriente de valores primitivos de Java. Se llena y lee de forma secuencial.

Correlación
Un conjunto de pares nombre-valor, donde los nombres son series y los valores son tipos primitivos de Java. Se puede acceder a las entradas secuencialmente o de forma aleatoria por el nombre. El orden de las entradas no está definido.

Texto
Un mensaje que contiene un java.lang.String.

Objeto
Un mensaje que contiene un objeto serializable Java.

Bytes
Una corriente de datos de bytes no interpretados. Este tipo de mensaje sirve para codificar literalmente un cuerpo para que coincida con un formato de mensaje existente.

El campo de cabecera JMSCorrelationID se utiliza para enlazar un mensaje con otro. Generalmente, enlaza un mensaje de respuesta con su mensaje de petición. JMSCorrelationID puede mantener el ID de mensaje específico de un proveedor, una serie específica de la aplicación o un valor de byte[] nativo del proveedor.

Selectores de mensajes en JMS
Los mensajes pueden contener valores de propiedad definidos por la aplicación. Una aplicación puede utilizar selectores de mensajes para hacer que un proveedor JMS filtre los mensajes.

Un mensaje contiene un recurso incorporado para dar soporte a valores de propiedad definidos por la aplicación. De hecho, esto proporciona un mecanismo que permite añadir campos de cabecera específicos de la aplicación a un mensaje. Las propiedades permiten que una aplicación, utilizando selectores de mensajes, haga que un proveedor JMS seleccione o filtre mensajes en nombre de la aplicación, utilizando criterios específicos de la aplicación. Las propiedades definidas por la aplicación deben cumplir las reglas siguientes:

- Los nombres de propiedad deben cumplir las normas de un identificador de selector de mensajes.
- Los valores de propiedad pueden ser de tipo booleano, byte, short, int, long, float, double y String.
- Los prefijos de nombre JMSX y JMS_ están reservados.

Los valores de propiedad se establecen antes de enviar un mensaje. Cuando un cliente recibe un mensaje, las propiedades del mensaje son de sólo lectura. Si un cliente intenta establecer propiedades en este punto, se emite una MessageNotWriteableException. Si se invoca clearProperties, las propiedades pueden ser de lectura y escritura.

Un valor de propiedad puede duplicar un valor en un cuerpo de mensaje. JMS no define una política para lo que se puede convertir en una propiedad. Pero los desarrolladores de aplicaciones deben tener en cuenta que los proveedores JMS probablemente manejan los datos de un cuerpo de mensajes de forma más eficiente que los datos de las propiedades de mensaje. Para obtener el mejor rendimiento, las aplicaciones deben utilizar propiedades de mensaje sólo cuando necesiten personalizar una cabecera de mensaje. La razón principal de hacer esto es permitir la selección personalizada de mensajes.

Un selector de mensajes de JMS permite que un cliente especifique los mensajes en los que está interesado utilizando la cabecera del mensaje. Sólo se entregan los mensajes cuyas cabeceras coinciden con el selector.

Los selectores de mensajes no pueden hacer referencia a valores de cuerpo de mensaje.

Un selector de mensajes coincide con un mensaje cuando la evaluación del selector da un resultado verdadero cuando el campo de cabecera de mensaje y los valores de propiedad se sustituyen por sus identificadores correspondientes en el selector.

Un selector de mensajes es una serie, cuya sintaxis se basa en un subconjunto de la sintaxis de expresión condicional SQL92. El orden en el que se evalúa un selector de mensajes es de izquierda a derecha dentro de un nivel de prioridad. Se pueden utilizar paréntesis para cambiar este orden. Los literales de selector y nombres de operador predefinidos se presentan aquí escritos en mayúsculas, pero no hay distinción entre mayúsculas y minúsculas.

Contenido de un selector de mensajes

Un selector de mensajes puede contener:

- Literales

  - Un literal de tipo serie encerrado entre comillas simples. Una comilla doble dentro del literal representa una comilla simple. Ejemplos: 'literal' y 'literal''s'. Como en el caso de los literales de tipo serie de Java, se utiliza la codificación de caracteres Unicode.
  - Un literal numérico exacto es un valor numérico sin coma decimal, tal como 57, -957 y +62. Se pueden utilizar números dentro del rango long de Java.
  - Un literal numérico aproximado es un valor numérico expresado en notación científica, tal como 7E3 o -57.9E2, o un valor numérico con un decimal, tal como 7, -95,7 o +6,2. Se pueden utilizar números dentro del rango double de Java.
  - Los literales booleanos TRUE y FALSE.
- Identificadores:
  - Un identificador es una secuencia de longitud ilimitada de letras Java y dígitos Java, el primero de los cuales debe ser una letra Java. Una letra es cualquier carácter para el que el método Character.isJavaLetter devuelve true. Esto incluye _ y $. Una letra o dígito es cualquier carácter para el que el método Character.isJavaLetterOrDigit devuelve true.
  - Los nombres NULL, TRUE o FALSE no pueden ser identificadores.
  - NOT, AND, OR, BETWEEN, LIKE, IN o IS no pueden ser identificadores.
  - Los identificadores son referencias de campo de cabecera o referencias de propiedad.
  - Los identificadores distinguen entre mayúsculas y minúsculas.
  - Las referencias de campo de cabecera de mensaje están restringidas a:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    Los valores JMSMessageID, JMSTimestamp, JMSCorrelationID y JMSType pueden ser nulos y, en este caso, se tratan como un valor NULL.
  - Cualquier nombre que empiece por JMSX es un nombre de propiedad definido por JMS.
  - Cualquier nombre que empiece por JMS_ es un nombre de propiedad específico del proveedor.
  - Cualquier nombre que no empiece por JMS es un nombre de propiedad específico de la aplicación. Si hay alguna referencia a una propiedad que no exista en un mensaje, su valor es NULL. Si existe, su valor es el de la propiedad correspondiente.
- El espacio en blanco equivale al que se ha definido para Java: espacio, separador horizontal, alimentación de papel y terminador de línea.
- Expresiones:
  - Un selector es una expresión condicional. Un selector cuya evaluación da un resultado verdadero produce una coincidencia; un selector cuya evaluación da un resultado falso o desconocido no produce una coincidencia.
  - Las expresiones aritméticas se componen de sí mismas, operaciones aritméticas, identificadores (con un valor que se trata como literal numérico) y literales numéricos.
  - Las expresiones condicionales se componen de sí mismas, operaciones de comparación y operaciones lógicas.
- Están permitidos los corchetes estándar () para establecer el orden en el que se evalúan las expresiones.
- Operadores lógicos por orden de prioridad: NOT, AND y OR.
- Operadores de comparación: =, >, >=, <, <=, <> (no igual).

  - Sólo se pueden comparar valores del mismo tipo, con la excepción de que se pueden comparar valores numéricos exactos y valores numéricos aproximados. (La conversión de tipo necesaria se define mediante las reglas de la promoción numérica de Java.) Si se intentan comparar tipos diferentes, el selector siempre es falso (false).
  - La comparación de serie y booleano está restringida a = y < >. Dos series son iguales sólo si contienen la misma secuencia de caracteres.
- Operadores aritméticos por orden de prioridad:
  - +, - unario.
  - *, /, multiplicación y división.
  - +, -, suma y resta.
  - No están permitidas las operaciones aritméticas sobre un valor NULL. Si se intentan, el selector completo siempre es falso.
  - Las operaciones aritméticas deben utilizar la promoción numérica de Java.
- Operador de comparación expr-aritm1 [NOT] BETWEEN expr-aritm2 y expr-aritm3:
  - Edad BETWEEN (entre) 15 y 19 es equivalente a edad >= 15 AND edad <= 19.
  - Edad NOT BETWEEN (no entre) 15 y 19 es equivalene a edad < 15 OR edad> 19.
  - Si alguna de las expresiones de una operación BETWEEN es NULL, el valor de la operación es falso (false). Si alguna de las expresiones de una operación NOT BETWEEN es NULL, el valor de la operación es verdadero (true).
- Operador de comparación identificador [NOT] IN (literal-serie1, literal-serie2,...), donde el identificador tiene un valor de serie o NULL.
  - País IN (' ReinoUnido', 'EEUU', 'Francia') es verdadero para 'ReinoUnido' y falso para 'Perú'. Equivale a la expresión (País = 'ReinoUnido') OR (País = 'EEUU') OR (País = 'Francia').
  - País NOT IN ('ReinoUnido', 'EEUU', 'Francia') es falso para 'EEUU' y verdadero para 'Perú'. Equivale a la expresión NOT ((País = 'ReinoUnido') OR (País = 'EEUU') OR (País = 'Francia')).
  - Si el identificador de una operación IN o NOT IN es NULL, el valor de la operación es desconocido (unknown).
- Operador de comparación identificador [NOT] valor de patrón LIKE, carácter de escape [ESCAPE], en el que el identificador tiene un valor de serie, el valor de patrón es un literal de serie, donde _ representa cualquier carácter y % representa cualquier secuencia de caracteres (incluida la secuencia vacía). Todos los demás caracteres se representan a sí mismos. El carácter de escape opcional es un literal de tipo serie, formado por un solo carácter, que se utiliza para invalidar el significado especial de _ y % en patrón-valor.
  - phone LIKE '12%3' es verdadero para 123 y 12993, y falso para 1234.
  - word LIKE 'l_se' es verdadero para "lose" y falso para "loose".
  - underscored LIKE '_%' ESCAPE '\' es verdadero para "_foo" y falso para "bar".
  - phone NOT LIKE '12%3' es falso para 123 y 12993, y verdadero para 1234.
  - Si el identificador de una operación LIKE o NOT LIKE es NULL, el valor de la operación es desconocido.
- El operador de comparación identificador IS NULL comprueba un valor de campo de cabecera nulo o un valor de propiedad no encontrado.
  - nombre_prop IS NULL.
- El operador de comparación identificador IS NOT NULL comprueba la existencia de un valor de campo de cabecera no nulo o un valor de propiedad.
  - nombre_prop IS NOT NULL.

Ejemplo de un selector de mensajes

El selector de mensajes siguiente selecciona mensajes con un tipo de mensaje de vehículo, color azul y de peso superior a 2500 libras:

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

Valores de propiedad NULL

Tal como se ha indicado anteriormente, los valores de propiedad pueden ser NULL. La semántica SQL 92 NULL define la evaluación de expresiones de selector que contengan valores NULL. La lista siguiente proporciona una breve descripción de estas semánticas:

- SQL trata un valor NULL como desconocido (unknown).
- El resultado de una comparación o aritmética con un valor desconocido siempre es un valor desconocido.
- El operador IS NULL convierte un valor desconocido en un valor TRUE.
- El operador IS NOT NULL convierte un valor desconocido en un valor FALSE.

Comportamiento especial de JMSMessageID y JMSCorrelationID

Las clases IBM MQ para JMS contienen optimizaciones al seleccionar mensajes de una cola basándose en JMSMessageID o JMSCorrelationID.

Si una aplicación especifica un selector con el formato:

JMSMessageID= 'ID:*ID_mensaje*'

donde *ID_mensaje* es un valor String que contiene un identificador de mensaje IBM MQ estándar y, a continuación, las clases IBM MQ para JMS utilizan **MatchOption** MQMO_MATCH_MSG_ID para obtener el mensaje con el identificador de mensaje especificado.

Por ejemplo, para obtener un mensaje con el identificador de mensaje 414D51207061756C745639314C545320C57C1A5F25ECE602 de una cola, una aplicación debe utilizar el selector de mensajes siguiente:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

De forma similar, si la aplicación especifica un selector que tiene el formato:

JMSCorrelationID = 'ID:*ID_correlación*'

donde *correlation_id* es una serie que contiene un identificador de correlación IBM MQ estándar, las clases IBM MQ para JMS utilizan **MatchOption** MQMO_MATCH_CORREL_ID para obtener el mensaje con el identificador de correlación especificado de la cola.

En el ejemplo siguiente, se utiliza un selector de mensajes para obtener un mensaje que tiene un identificador de correlación de 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Si un selector de mensajes contiene un valor de todos los ceros para *message_id* o *correlation_id*, coincide con cualquier mensaje de la cola. Por ejemplo, si una aplicación utiliza el selector:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

entonces cualquier mensaje en la cola se considera una coincidencia y se devuelve a la aplicación.

Para obtener más información sobre MQMO_MATCH_MSG_ID y MQMO_MATCH_CORREL_ID **MatchOptions**, consulte MatchOptions (MQLONG).

Restricciones

Aunque SQL permite la comparación y la aritmética números decimales fijos, los selectores de mensajes de JMS no lo permiten. Por ello, los literales numéricos exactos están restringidos a los que no tienen ningún decimal. Esto también es el motivo por el que hay numerales con un decimal como representación alternativa para un valor numérico aproximado.

No se pueden utilizar comentarios de SQL.

Correlación de mensajes JMS en mensajes IBM MQ
Los mensajes de IBM MQ se componen de un Descriptor de mensaje, una cabecera MQRFH2 opcional y un cuerpo. El contenido de un mensaje de JMS se correlaciona y copia parcialmente en un mensaje de IBM MQ.

Este tema describe cómo la estructura de mensajes de JMS que se describe en la primera parte de esta sección se correlaciona con un mensaje de IBM MQ. Es de interés para los programadores que deseen transmitir mensajes entre JMS y aplicaciones de IBM MQ tradicionales. También es de interés para los usuarios que deseen manejar mensajes transmitidos entre dos aplicaciones de JMS, por ejemplo, en una implementación de IBM Integration Bus.

La información de esta sección no es aplicable si una aplicación utiliza una conexión en tiempo real con un intermediario. Cuando una aplicación utiliza una conexión en tiempo real, toda la comunicación se lleva a cabo directamente sobre TCP/IP; no intervienen colas ni mensajes de IBM MQ.

Los mensajes de IBM MQ constan de tres componentes:

- El descriptor de mensaje de IBM MQ (MQMD)
- Una cabecera MQRFH2 de IBM MQ
- El cuerpo del mensaje.

La cabecera MQRFH2 es opcional, y su inclusión en un mensaje de salida está determinada por el distintivo TARGCLIENT en la clase de destino de JMS. Puede establecer este distintivo utilizando la herramienta de administración de IBM MQ JMS. Debido a que la cabecera MQRFH2 transporta información específica de JMS, inclúyala siempre en el mensaje cuando el emisor sepa que el destino de recepción es una aplicación de JMS. Normalmente, la cabecera MQRFH2 se omite cuando se envía un mensaje directamente a una aplicación que no es de JMS. Esto se debe a que una aplicación de este tipo no espera una cabecera MQRFH2 en su mensaje de IBM MQ.

Si un mensaje entrante no tiene una cabecera MQRFH2, el objeto Queue o Topic que se deriva del campo de cabecera JMSReplyTo del mensaje, de forma predeterminada, tiene este distintivo establecido, de forma que un mensaje de respuesta que se envía a la cola o al tema tampoco tiene una cabecera MQRFH2. Puede desactivar este comportamiento de incluir una cabecera MQRFH2 en un mensaje de respuesta solo si el mensaje original tiene una cabecera MQRFH2. Para ello establezca la propiedad TARGCLIENTMATCHING de la fábrica de conexiones en NO.

La Figura 10 en la página 154 muestra cómo la estructura de un mensaje de JMS se transforma en un mensaje de IBM MQ y luego el proceso inverso:

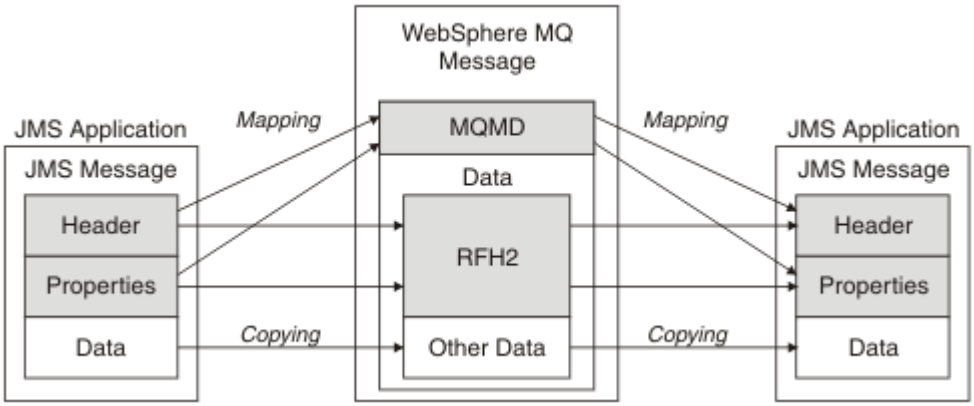


Figura 10. Cómo los mensajes entre JMS y IBM MQ se transforman utilizando la cabecera MQRFH2.

Las estructuras se transforman de dos modos:

Correlación
Cuando el MQMD incluye un campo que es equivalente al campo de JMS, el campo de JMS se correlaciona con el campo de MQMD. Se exponen campos MQMD adicionales como propiedades de JMS, porque una aplicación JMS podría necesitar obtener o establecer estos campos cuando se comunique con una aplicación no JMS.

Copia
Cuando no existe ningún equivalente de MQMD, se pasa una propiedad o campo de cabecera de JMS, posiblemente transformada, como un campo dentro de MQRFH2.

La cabecera MQRFH2 y JMS
Esta colección de temas describe la cabecera MQRFH Versión 2, que transporta datos específicos de JMS que están asociados con el contenido del mensaje. La cabecera MQRFH Versión 2 es ampliable y también puede transportar información adicional que no está asociada directamente con JMS. Pero la presente sección sólo describe su utilización por parte de JMS.Para obtener una descripción completa, consulte MQRFH2 - Reglas y cabecera de formato 2.

La cabecera consta de dos partes, una parte fija y otra variable.

Parte fija
La parte fija se construye de acuerdo con el patrón de cabecera *estándar* de IBM MQ y consta de los campos siguientes:

StrucId (MQCHAR4)
Identificador de estructura.

Debe ser MQRFH_STRUC_ID (valor: "RFH ") (valor inicial).

MQRFH_STRUC_ID_ARRAY también se define (valor: "R", "F", "H", " ").

Versión (MQLONG)
Número de versión de la estructura.

Debe ser MQRFH_VERSION_2 (valor: 2) (valor inicial)

StrucLength (MQLONG)
Longitud total de MQRFH2, incluidos los campos NameValueData.

El valor establecido en StrucLength debe ser un múltiplo de 4 (para ello, los datos de los campos NameValueData se pueden rellenar con caracteres de espacio).

Encoding (MQLONG)
Codificación de datos.

Codificación de todos los datos numéricos contenidos en la parte del mensaje que sigue a continuación de MQRFH2 (la cabecera siguiente o los datos de mensaje que siguen a esta cabecera).

CodedCharSetId (MQLONG)
Identificador de juego de caracteres codificados.

Representación de todos los datos de tipo carácter contenidos en la parte del mensaje que sigue a MQRFH2 (la cabecera siguiente o los datos de mensaje que siguen a esta cabecera).

Format (MQCHAR8)
Nombre del formato.

Nombre del formato de la parte del mensaje que sigue a MQRFH2.

Flags (MQLONG)
Distintivos.

MQRFH_NO_FLAGS = 0. No se han establecido distintivos.

NameValueCCSID (MQLONG)

CCSID (identificador de juego de caracteres codificados) para las series NameValueData contenidas en esta cabecera. NameValueData se puede codificar en un juego de caracteres que difiera de las demás series contenidas en la cabecera (StrucID y Format).

Si NameValueCCSID es un CCSID Unicode de 2 bytes (1200, 13488 o 17584), el orden de bytes de Unicode es el mismo que el orden de bytes de los campos numéricos de MQRFH2. (Por ejemplo, Version, StrucLength y NameValueCCSID).

Tabla 16. Valores posibles para el campo NameValueCCSID

CCSID	**Significado**
1200	UTF-16, versión soportada más reciente de Unicode
13488	UTF-16, subconjunto de Unicode versión 2.0
17584	UTF-16, subconjunto de Unicode versión 3.0 (incluye símbolo del Euro)
1208	UTF-8, versión soportada más reciente de Unicode

Parte variable

La parte variable sigue a la parte fija. Esta parte contiene un número variable de carpetas MQRFH2. Cada carpeta contiene un número variable de elementos o propiedades. Propiedades relacionadas con el grupo Carpetas. Las cabeceras MQRFH2 creadas por JMS pueden contener cualquiera de las carpetas siguientes:

Carpeta `mcd`

`mcd` contiene propiedades que describen el formato del mensaje. Por ejemplo, la propiedad de dominio de servicio de mensajes `Msd` identifica un mensaje JMS como `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` o nulo.

La carpeta `mcd` siempre está presente en un mensaje JMS que contiene un `MQRFH2`.

Siempre está presente en un mensaje que contiene un `MQRFH2` enviado desde IBM Integration Bus. Describe el dominio, formato, tipo y conjunto de mensajes de un mensaje.

Tabla 17. mcd nombre de propiedad, sinónimo, tipo de datos y carpeta

Sinónimo de propiedad	**Nombre de propiedad**	**Tipo de datos**	**Carpeta**
	`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>`

No añada sus propias propiedades en la carpeta `mcd`.

Carpeta `jms`

`jms` contiene campos de cabecera JMS y propiedades JMSX que no se pueden expresar completamente en `MQMD`. La carpeta `jms` siempre está presente en un JMS `MQRFH2`.

Carpeta `usr`

`usr` contiene propiedades JMS definidas por la aplicación asociadas al mensaje. La carpeta `usr` solo está presente si una aplicación ha establecido una propiedad definida por la aplicación.

Carpeta `mqext`

`mqext` contiene los siguientes tipos de propiedad:

- Propiedades que únicamente utiliza WebSphere Application Server.
- Propiedades relacionadas con la entrega retardada de mensajes.

La carpeta está presente si la aplicación ha establecido como mínimo una de las propiedades definidas por IBM o ha utilizado el retardo el retardo de entrega.

Tabla 18. `mqext` nombre de propiedad, sinónimo, tipo de datos y carpeta

Sinónimo de propiedad	Nombre de propiedad	Tipo de datos	Carpeta
`JMSArmCorrelator`	`mqext.Arm`	`string`	`<mqext><Arm>`*`armCorrelator`*`</Arm></mqext>`
`JMSRMCorrelator`	`mqext.Wrm`	`string`	`<mqext><Wrm>`*`wrmCorrelator`*`</Wrm></mqext>`
`JMSDeliveryTime`	`mqext.Dlt`	`i8`	`<mqext><Dlt>`*`DeliveryTime`*`</Dlt></mqext>`
`JMSDeliveryDelay`	`mqext.Dly`	`i8`	`<mqext><Dly>`*`DeliveryTime`*`</Dly></mqext>`

No añada sus propias propiedades en la carpeta `mqext`.

Carpeta `mqps`

`mqps` contiene propiedades que solo son utilizadas por la publicación/suscripción de IBM MQ. La carpeta sólo está presente si la aplicación ha establecido al menos una de las propiedades de publicación/suscripción integradas.

Tabla 19. `mqps` nombre de propiedad, sinónimo, tipo de datos y carpeta

Sinónimo de propiedad	Nombre de propiedad	Tipo de datos	Carpeta
`MQTopicString`	`mqps.Top`	`string`	`<mqps><Top>`*`topicString`*`</Top></mqps>`
`MQSubUserData`	`mqps.Sud`	`string`	`<mqps><Sud>`*`subscriberUserData...`*`</Sud></mqps>`
`MQIsRetained`	`mqps.Ret`	`boolean`	`<mqps><Ret>`*`isRetained`*`</Ret></mqps>`
`MQPubOptions`	`mqps.Pub`	`i8`	`<mqps><Pub>`*`publicationOptions`*`</Pub></mqps>`
`MQPubLevel`	`mqps.Pbl`	`i8`	`<mqps><Pbl>`*`publicationLevel`*`</Pbl></mqps>`
`MQPubTime`	`mqpse.Pts`	`string`	`<mqps><Pts>`*`publicationTime`*`</Pts></mqps>`
`MQPubSeqNum`	`mqpse.Seq`	`i8`	`<mqps><Seq>`*`publicationSequenceNumber`*`</Seq></mqps>`
`MQPubStrIntData`	`mqpse.Sid`	`string`	`<mqps><Sid>`*`publicationData`*`</Sid></mqps>`

Tabla 19. mqps nombre de propiedad, sinónimo, tipo de datos y carpeta (continuación)			
Sinónimo de propiedad	**Nombre de propiedad**	**Tipo de datos**	**Carpeta**
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

No añada sus propias propiedades en la carpeta mqps.

En la Tabla 20 en la página 158 se muestra una lista completa de los nombres de las propiedades.

Tabla 20. Carpetas y propiedades de MQRFH2 utilizadas por JMS				
Nombre de campo de JMS	**Tipo de Java**	**Nombre de carpeta de MQRFH2**	**Nombre de propiedad**	**Tipo/valores**
JMSDestination	Destino	jms	Dst	serie
JMSExpiration	Entero largo	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Serie	jms	Cid	serie
JMSReplyTo	Destino	jms	Rto	serie
JMSTimestamp	Entero largo	jms	Tms	i8
JMSType	Serie	mcd	Type, Set, Fmt	serie
JMSXGroupID	Serie	jms	Gid	serie
JMSXGroupSeq	int	jms	Seq	i4
xxx (definido usuario)	Cualquiera	usr	xxx	cualquiera
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
Longitud en bytes de la serie NameValueData que sigue inmediatamente a este campo de longitud (no incluye su longitud propia).

NameValueData (MQCHARn)
Serie de un solo carácter, para la que el campo NameValueLength anterior establece la longitud en bytes. Contiene una carpeta con una secuencia de propiedades. Cada propiedad es un trío de nombre/tipo/valor contenido en un elemento XML cuyo nombre es el nombre de la carpeta, tal como se muestra a continuación:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

El código </foldername> de cierre puede ir seguido de espacios como caracteres de relleno. Cada triplete se codifica utilizando una sintaxis similar a la de XML:

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

El elemento dt='datatype' es opcional y se omite para muchas propiedades, ya que el tipo de datos está predefinido. Si se incluye, se deben añadir uno o más caracteres de espacio antes del código dt=.

name
es el nombre de la propiedad. Consulte la Tabla 20 en la página 158.

datatype
debe coincidir, después de agrupar los datos, con uno de los tipos de datos listados en la Tabla 21 en la página 159.

value
es una representación de tipo serie del valor que se debe transmitir, utilizando las definiciones de la Tabla 21 en la página 159.

Un valor nulo se codifica utilizando la sintaxis siguiente:

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

No utilice xsi:nil='false'.

Tabla 21. Tipos de datos de propiedad

Tipo de datos	**Definición**
serie	Cualquier secuencia de caracteres, excepto < y &
boolean	El carácter 0 ó 1 (0 = false,1 = true)
bin.hex	Dígitos hexadecimales que representan octetos
i1	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -128 a 127 inclusive
i2	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -32768 a 32767 inclusive
i4	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -2147483648 a 2147483647 inclusive
i8	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -9223372036854775808 a 9223372036854775807 inclusive
int	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del mismo rango que i8. Se puede utilizar en lugar de los tipos i* si el emisor no desea asociar una precisión específica a una propiedad
r4	Número de coma flotante, magnitud < = 3.40282347E+38, > = 1.175E-37 expresado utilizando dígitos 0..9, signo opcional, dígitos fraccionarios opcionales, exponente opcional
r8	Número de coma flotante, magnitud < = 1.7976931348623E+308, > = 2.225E-307 expresado utilizando dígitos 0..9, signo opcional, dígitos fraccionarios opcionales, exponente opcional

Un valor de serie puede contener espacios. En un valor de serie, debe utilizar las secuencias de escape siguientes:

- `&` para el carácter &
- `<` para el carácter <

Puede utilizar las secuencias de escape siguientes, pero no son obligatorias:

- `>` para el carácter >
- `'` para el carácter '
- `"` para el carácter "

Campos y propiedades de JMS y sus campos correspondientes de MQMD
Estas tablas muestran los campos de MQMD que son equivalentes a campos de cabecera de JMS, propiedades de JMS y propiedades específicas del proveedor de JMS

La Tabla 22 en la página 160 lista los campos de cabecera de JMS y la Tabla 23 en la página 160 lista las propiedades de JMS que se correlacionan directamente con campos de MQMD. La Tabla 24 en la página 160 lista las propiedades específicas del proveedor y los campos de MQMD con los que se correlacionan.

Tabla 22. Correlación de campos de cabecera de JMS con campos de MQMD

Campo de cabecera de JMS	**Tipo de Java**	**Campo de MQMD**	**Tipo C**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	Entero largo	Caducidad	MQLONG
JMSPriority	int	Prioridad	MQLONG
JMSMessageID	Serie	MsgID	MQBYTE24
JMSTimestamp	Entero largo	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Serie	CorrelId	MQBYTE24

Tabla 23. Correlación de propiedades de JMS con campos de MQMD

Propiedad de JMS	**Tipo de Java**	**Campo de MQMD**	**Tipo C**
JMSXUserID	Serie	UserIdentifier	MQCHAR12
JMSXAppID	Serie	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Serie	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabla 24. Correlación de propiedades específicas del proveedor de JMS con campos de MQMD

Propiedad específica del proveedor de JMS	**Tipo de Java**	**Campo de MQMD**	**Tipo C**
JMS_IBM_Report_Exception	int	Informe	MQLONG
JMS_IBM_Report_Expiration	int	Informe	MQLONG
JMS_IBM_Report_COA	int	Informe	MQLONG

Tabla 24. Correlación de propiedades específicas del proveedor de JMS con campos de MQMD (continuación)

Propiedad específica del proveedor de JMS	Tipo de Java	Campo de MQMD	Tipo C
JMS_IBM_Report_COD	int	Informe	MQLONG
JMS_IBM_Report_PAN	int	Informe	MQLONG
JMS_IBM_Report_NAN	int	Informe	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Informe	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Informe	MQLONG
JMS_IBM_Report_Discard_Msg	int	Informe	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Comentarios	MQLONG
JMS_IBM_Format	Serie	Format "1" en la página 161	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	Serie	CodedCharacterSetId "2" en la página 161	MQLONG
JMS_IBM_PutDate	Serie	PutDate	MQCHAR8
JMS_IBM_PutTime	Serie	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

Nota:

1. JMS_IBM_Format representa el formato del cuerpo del mensaje. Esto se puede ser definido por la aplicación estableciendo la propiedad JMS_IBM_Format del mensaje (observe que hay un límite de 8 caracteres), o puede tomar como valor predeterminado el formato de IBM MQ de cuerpo de mensaje adecuado al tipo de mensaje de JMS. JMS_IBM_Format sólo se correlaciona con el campo Format de MQMD si el mensaje no contiene secciones RFH o RFH2. En un mensaje típico, se correlaciona con el campo Format de la cabecera RFH2 que precede inmediatamente al cuerpo del mensaje.
2. El valor de la propiedad JMS_IBM_Character_Set es un valor de tipo serie que contiene el conjunto de caracteres Java equivalente para el valor numérico CodedCharacterSetId. El campo CodedCharacterSetId de MQMD es un valor numérico que contiene el equivalente del juego de caracteres de Java especificado por la propiedad JMS_IBM_Character_Set.

Correlación de campos de JMS con campos de IBM MQ (mensajes salientes)
Estas tablas muestran cómo los campos de cabecera y de propiedad de JMS se correlacionan con campos de MQMD y MQRFH2 al ejecutar las operaciones send() o publish().

La Tabla 25 en la página 162 muestra cómo se correlacionan los campos de cabecera de JMS con los campos de MQMD/RFH2 en las operaciones send() o publish(). La Tabla 26 en la página 162 muestra cómo se correlacionan las propiedades de JMS con los campos de MQMD/RFH2 en las operaciones send() o publish(). La Tabla 27 en la página 163 muestra cómo se correlacionan las propiedades específicas del proveedor de JMS con los campos de MQMD en las operaciones send() o publish().

Para los campos con la indicación "Establecido por Objeto de mensaje", el valor transmitido es el valor contenido en el mensaje de JMS inmediatamente antes de la operación send () o publish (). El valor contenido en el mensaje de JMS no es alterado por la operación.

Para los campos con la indicación "Establecido por Método Send", se asigna un valor cuando se ejecuta send() o publish() (no se tiene en cuenta el valor contenido en el mensaje de JMS). El valor contenido en el mensaje de JMS se actualiza para mostrar el valor utilizado.

Los campos con la indicación "Sólo recepción" no se transmiten y su contenido no es alterado por send() o publish().

Tabla 25. Correlación de los campos de mensajes salientes

Nombre del campo de cabecera de JMS	Campo de MQMD utilizado para la transmisión	Cabecera	Establecido por
JMSDestination		MQRFH2	Método Send
JMSDeliveryMode	Persistence	MQRFH2	Método Send
JMSExpiration	Caducidad	MQRFH2	Método Send
JMSPriority	Prioridad	MQRFH2	Método Send
JMSMessageID	MsgID		Método Send
JMSTimestamp	PutDate/PutTime		Método Send
JMSCorrelationID	CorrelId	MQRFH2	Objeto de mensaje
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Objeto de mensaje
JMSType		MQRFH2	Objeto de mensaje
JMSRedelivered			Sólo recepción

Nota:

1. El campo CodedCharacterSetId de MQMD es un valor numérico que contiene el equivalente del juego de caracteres de Java especificado por la propiedad JMS_IBM_Character_Set.

Tabla 26. Correlación de propiedades de mensajes salientes de JMS

Nombre de propiedad de JMS	Campo de MQMD utilizado para la transmisión	Cabecera	Establecido por
JMSXUserID	UserIdentifier		Método Send
JMSXAppID	PutApplName		Método Send
JMSXDeliveryCount			Sólo recepción
JMSXGroupID	GroupId	MQRFH2	Objeto de mensaje
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Objeto de mensaje

Nota:

Estas propiedades son definidas como de solo lectura por la especificación JMS y son establecidas (en algunos casos, opcionalmente) por el proveedor JMS.

En IBM MQ classes for JMS, dos de estas propiedades se pueden alterar temporalmente mediante la aplicación. Para ello, asegúrese de que el destino se ha configurado apropiadamente estableciendo las propiedades siguientes:

1. Establezca la propiedad `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` en `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Establezca la propiedad `WMQConstants.WMQ_MQMD_WRITE_ENABLED` en `true`.

Las propiedades siguientes se pueden alterar temporalmente mediante la aplicación:

JMSXAppID
Esta propiedad se puede alterar temporalmente estableciendo la propiedad `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` en el mensaje-el valor debe ser una serie Java .

JMSXGroupID
Esta propiedad se puede alterar temporalmente estableciendo la propiedad `WMQConstants.JMS_IBM_MQMD_GROUPID` en el mensaje, el valor debe ser una matriz de bytes.

Tabla 27. Correlación de propiedades específicas del proveedor de mensajes salientes de JMS

Nombre de propiedad específica del proveedor de JMS	Campo de MQMD utilizado para la transmisión	Cabecera	Establecido por
JMS_IBM_Report_Exception	Informe		Objeto de mensaje
JMS_IBM_Report_Expiration	Informe		Objeto de mensaje
JMS_IBM_Report_COA/COD	Informe		Objeto de mensaje
JMS_IBM_Report_NAN/PAN	Informe		Objeto de mensaje
JMS_IBM_Report_Pass_Msg_ID	Informe		Objeto de mensaje
JMS_IBM_Report_Pass_Correl_ID	Informe		Objeto de mensaje
JMS_IBM_Report_Discard_Msg	Informe		Objeto de mensaje
JMS_IBM_MsgType	MsgType		Objeto de mensaje
JMS_IBM_Feedback	Comentarios		Objeto de mensaje
JMS_IBM_Format	Formato		Objeto de mensaje
JMS_IBM_PutApplType	PutApplType		Método Send
JMS_IBM_Encoding	Encoding		Objeto de mensaje
JMS_IBM_Character_Set	CodedCharacterSetId		Objeto de mensaje
JMS_IBM_PutDate	PutDate		Método Send
JMS_IBM_PutTime	PutTime		Método Send
JMS_IBM_Last_Msg_In_Group	MsgFlags		Objeto de mensaje

Correlación de campos de cabecera de JMS durante el envío o publicación
Estas notas están relacionadas con la correlación de los campos de JMS durante las operaciones de envío o publicación.

JMSDestination con MQRFH2
Esto se almacena como una serie que serializa las características salientes del objeto de destino para que un JMS receptor pueda reconstituir un objeto de destino equivalente. El campo MQRFH2 se codifica como URI (consulte “Identificadores uniformes de recursos (URI)” en la página 228 para conocer detalles sobre la notación URI).

JMSReplyTo con MQMD.ReplyToQ, ReplyToQMgr, MQRFH2
El nombre de cola se copia en el campo MQMD.ReplyToQ, y el nombre del gestor de colas se copia en los campos ReplyToQMgr. La información de la extensión de destino se copia en el campo MQRFH2 (otros detalles útiles se mantienen en el objeto de destino). El campo MQRFH2 se codifica como URI (consulte “Identificadores uniformes de recursos (URI)” en la página 228 para conocer detalles sobre la notación URI).

JMSDeliveryMode con MQMD.Persistence
MessageProducer o el método send() o publish() establecen el valor JMSDeliveryMode, a menos que lo altere temporalmente el objeto de destino. El valor JMSDeliveryMode se correlaciona con el campo MQMD.Persistence tal como se indica a continuación:

- El valor PERSISTENT de JMS es equivalente a MQPER_PERSISTENT
- El valor NON_PERSISTENT de JMS es equivalente a MQPER_NOT_PERSISTENT

Si la propiedad de persistencia de MQQueue no se establece en WMQConstants.WMQ_PER_QDEF, el valor de la modalidad de entrega también se codifica en MQRFH2.

JMSExpiration con MQMD.Expiry, MQRFH2
JMSExpiration almacena el tiempo de caducidad (la suma de la hora actual y el tiempo de vida), mientras que MQMD almacena el tiempo de vida. Además, JMSExpiration está en milisegundos, pero MQMD.Expiry es la décima parte de un segundo.

- Si el método send() establece un tiempo de vida ilimitado, MQMD.Expiry se establece en MQEI_UNLIMITED y no se codifica ninguna JMSExpiration en MQRFH2.
- Si el método send() establece un tiempo de vida inferior a 214748364.7 segundos (7 años, aproximadamente), el tiempo de vida se almacena en MQMD.Expiry y la hora de caducidad (en milisegundos), se codifica como un valor i8 en MQRFH2.
- Si el método send() establece un tiempo de vida superior a 214748364.7 segundos, MQMD.Expiry se establece en MQEI_UNLIMITED. La hora de caducidad real en milisegundos se codifica como un valor i8 en MQRFH2.

JMSPriority con MQMD.Priority
Correlaciona directamente el valor JMSPriority (0-9) con el valor de prioridad MQMD (0-9). Si se establece JMSPriority en un valor que no sea el valor predeterminado, el nivel de prioridad también se codifica en MQRFH2.

JMSMessageID de MQMD.MessageID
Todos los mensajes enviados desde JMS tienen identificadores de mensaje exclusivos asignados por IBM MQ. El valor asignado se devuelve en el campo MQMD.MessageId después de la llamada de MQPUT y se vuelve a pasar a la aplicación en el campo JMSMessageID. El messageId de IBM MQ es un valor binario de 24 bytes, mientras que JMSMessageID es una serie de caracteres. JMSMessageID consta del valor messageId binario convertido en una secuencia de 48 caracteres hexadecimales con los caracteres ID: como prefijo. JMS proporciona una sugerencia que se puede establecer para inhabilitar la producción de identificadores de mensajes. Se pasa por alto esta sugerencia y se asigna un identificador exclusivo en todos los casos. Se sobrescribe cualquier valor especificado en el campo JMSMessageID antes de realizar una operación send().

Si necesita la capacidad para especificar el MQMD.MessageID, puede hacerlo con una de las extensiones de IBM MQ JMS descritas en “Lectura y escritura del descriptor de mensaje desde una aplicación de IBM MQ classes for JMS” en la página 251.

JMSTimestamp a MQRFH2
Durante un envío, el campo JMSTimestamp se establece de acuerdo con el reloj de la JVM. Este valor se establece en MQRFH2. Todos los valores que se establecen en el campo JMSTimestamp antes de realizar un envío (send()) se sobrescriben. Consulte también las propiedades JMS_IBM_PutDate y JMS_IBM_PutTime.

JMSType a MQRFH2
Esta serie se establece en el campo MQRFH2 mcd.Type. Si se encuentra en formato URI, también puede afectar a los campos mcd.Set y mcd.Fmt.

JMSCorrelationID a MQMD.CorrelId, MQRFH2
JMSCorrelationID puede mantener uno de los siguientes:

Un ID de mensaje específico del proveedor
Éste es el identificador de un mensaje enviado o recibido previamente, por lo que debe ser una serie de menos de 48 dígitos hexadecimales en minúscula con el prefijo `ID:`: el prefijo se elimina, los caracteres restantes se convierten en binarios y después se establecen en el campo MQMD.CorrelId.

Un valor byte[] nativo del proveedor
El valor se copia en el campo MQMD.CorrelId y se rellena con valores nulos o se trunca en 24 bytes si es necesario. En MQRFH2 no se codifica ningún valor CorrelId.

Una serie específica de la aplicación
El valor se copia en MQRFH2. Los primeros 24 bytes de la serie, en formato UTF8, se escriben en MQMD.CorrelID.

Correlación de campos de propiedad de JMS
Estas notas hacen referencia a la correlación de campos de propiedad de JMS contenidos en mensajes de IBM MQ.

JMSXUserID de UserIdentifier de MQMD
JMSXUserID se establece al finalizar la llamada de envío.

JMSXAppID de PutApplName de MQMD
JSMXAppID se establece al finalizar la llamada de envío.

JMSXGroupID a MQRFH2 (punto a punto)
Para mensajes punto a punto, JMSXGroupID se copia en el campo MQMD GroupID. Si JMSXGroupID empieza con el prefijo ID:, se convierte a binario. De lo contrario, se codifica como una serie de caracteres UTF8. Si es necesario, se rellena o se trunca el valor en la longitud de 24 bytes. Se establece el distintivo MQMF_MSG_IN_GROUP.

JMSXGroupID a MQRFH2 (publicación/suscripción)
Para los mensajes de publicación/suscripción, se copia el JMSXGroupID en MQRFH2 como una serie de caracteres.

JMSXGroupSeq MQMD MsgSeqNumber (punto a punto)
Para los mensajes punto a punto, se copia JMSXGroupSeq en el campo MsqSeqNumber de MQMD. Se establece el distintivo MQMF_MSG_IN_GROUP.

JMSXGroupSeq MQMD MsgSeqNumber (publicación/suscripción)
Para los mensajes de publicación/suscripción, se copia JMSXGroupSeq en MQRFH2 como i4.

Correlación de campos específicos del proveedor JMS
Las notas siguientes hacen referencia a la correlación de campos específicos del proveedor JMS con mensajes de IBM MQ.

JMS_IBM_Report_*XXX* con MQMD Report
Una aplicación JMS puede establecer las opciones de informe MQMD, utilizando las propiedades siguientes JMS_IBM_Report_*XXX*. El MQMD se puede correlacionar con varias propiedades JMS_IBM_Report_*XXX*.

Las constantes JMS_IBM_Report_*XXX* están en `com.ibm.msg.client.jakarta.wmq.WMQConstants` o `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_Report_Exception

MQRO_EXCEPTION o
MQRO_EXCEPTION_WITH_DATA o
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION o
MQRO_EXPIRATION_WITH_DATA o
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA o
MQRO_COA_WITH_DATA o
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD o
MQRO_COD_WITH_DATA o bien
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

Los valores MQRO están en `com.ibm.mq.constants.CMQC`.

JMS_IBM_MsgType a MQMD MsgType
El valor se correlaciona directamente con MQMD MsgType. Si la aplicación no ha establecido explícitamente un valor de JMS_IBM_MsgType, se utiliza el valor predeterminado, que se determina tal como se indica a continuación:

- Si JMSReplyTo se establece en un destino de cola de IBM MQ, MSGType se establece en el valor MQMT_REQUEST
- Si JMSReplyTo no está establecido o está establecido en algo que no sea un destino de cola de IBM MQ, MsgType se establece en el valor MQMT_DATAGRAM

JMS_IBM_Feedback con Feedback de MQMD
El valor se correlaciona directamente con Feedback de MQMD.

JMS_IBM_Format con Format de MQMD
El valor se correlaciona directamente con Format de MQMD.

JMS_IBM_Encoding con Encoding de MQMD
Si está establecida, esta propiedad altera temporalmente la codificación numérica de la cola de destino o tema.

JMS_IBM_Character_Set con CodedCharacterSetId de MQMD
Si está establecida, esta propiedad altera temporalmente la propiedad del juego de caracteres codificados de la cola de destino o tema.

JMS_IBM_PutDate a partir de PuTdate de MQMD
El valor de esta propiedad se establece, durante el envío, directamente desde el campo PutDate de MQMD. Todos los valores que se establecen en la propiedad JMS_IBM_PutDate antes de un envío se sobrescriben. Este campo es una serie de ocho caracteres, con el formato de fecha AAAAMMDD de IBM MQ. Esta propiedad se puede utilizar con la propiedad JMS_IBM_PutTime para determinar la hora en que se ha transferido el mensaje de acuerdo con el gestor de colas.

JMS_IBM_PutTime a partir de PutTime de MQMD
El valor de esta propiedad se establece, durante el envío, directamente desde el campo PutTime de MQMD. Todos los valores que se establecen en la propiedad JMS_IBM_PutTime antes de realizar un envío se sobrescriben. Este campo es una serie de ocho caracteres, con el formato de hora HHMMSSTH de IBM MQ. Esta propiedad se puede utilizar junto con la propiedad JMS_IBM_PutDate para determinar la hora en que se ha transferido el mensaje según el gestor de colas.

JMS_IBM_Last_Msg_In_Group con MsgFlags de MQMD
Para la mensajería punto a punto, este valor booleano se correlaciona con el distintivo MQMF_LAST_MSG_IN_GROUP del campo MsgFlags de MQMD. Normalmente se utiliza con las propiedades JMSXGroupID y JMSXGroupSeq para indicar a una aplicación heredada de IBM MQ que este mensaje es el último de un grupo. Esta propiedad no se tiene en cuenta para la mensajería de publicación/suscripción.

Correlación de campos de IBM MQ con campos de JMS (mensajes de entrada)
Estas tablas muestran cómo los campos de cabecera y de propiedad de JMS se correlacionan con campos de MQMD y MQRFH2 al ejecutar las operaciones get() o receive().

La Tabla 28 en la página 167 muestra cómo se correlacionan los campos de cabecera de JMS con los campos de MQMD/MQRFH2 al ejecutar las operaciones get() o receive(). La Tabla 29 en la página 168 muestra cómo se correlacionan los campos de propiedad de JMS con los campos de MQMD/MQRFH2 al ejecutar las operaciones get() o receive(). La Tabla 30 en la página 168 muestra cómo se correlacionan las propiedades de JMS específicas del proveedor.

Tabla 28. Correlación de los campos de cabecera de JMS para mensajes entrantes

Nombre del campo de cabecera de JMS	**Campo de MQMD recuperado de**	**Campo de MQRFH2 recuperado de**
JMSDestination		jms.Dst o mqps.Top "1" en la página 167
JMSDeliveryMode	Persistence "2" en la página 168	jms.Dlv "2" en la página 168
JMSExpiration		jms.Exp
JMSPriority	Prioridad	
JMSMessageID	MsgID	
JMSTimestamp	PutDate "2" en la página 168 PutTime "2" en la página 168	jms.Tms "2" en la página 168
JMSCorrelationID	CorrelId "2" en la página 168	jms.Cid "2" en la página 168
JMSReplyTo	ReplyToQ "2" en la página 168 ReplyToQMgr "2" en la página 168	jms.Rto "2" en la página 168
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Nota:

1. Si se definen jms.Dst y mqps.Top, se utiliza el valor contenido en jms.Dst.

2. Para las propiedades que pueden tener valores recuperados de MQRFH2 o de MQMD, si ambos están disponibles, se utiliza el valor de MQRFH2.
3. El valor de la propiedad JMS_IBM_Character_Set es un valor de tipo serie que contiene el conjunto de caracteres Java equivalente para el valor numérico CodedCharacterSetId.

Tabla 29. Correlación de propiedades para mensajes entrantes

Nombre de propiedad de JMS	**Campo de MQMD recuperado de**	**Campo de MQRFH2 recuperado de**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId "1" en la página 168	jms.Gid "1" en la página 168
JMSXGroupSeq	MsgSeqNumber "1" en la página 168	jms.Seq "1" en la página 168

Nota:

1. Para las propiedades que pueden tener valores recuperados de MQRFH2 o de MQMD, si ambos están disponibles, se utiliza el valor de MQRFH2. Las propiedades se establecen a partir de los valores MQMD sólo si se han establecido los parámetros de mensaje MQMF_MSG_IN_GROUP o MQMF_LAST_MSG_IN_GROUP.

Tabla 30. Correlación de propiedades de JMS específicas del proveedor para mensajes entrantes

Nombre de propiedad de JMS	**Campo de MQMD recuperado de**	**Campo de MQRFH2 recuperado de**
JMS_IBM_Report_Exception	Informe	
JMS_IBM_Report_Expiration	Informe	
JMS_IBM_Report_COA	Informe	
JMS_IBM_Report_COD	Informe	
JMS_IBM_Report_PAN	Informe	
JMS_IBM_Report_NAN	Informe	
JMS_IBM_Report_ Pass_Msg_ID	Informe	
JMS_IBM_Report_Pass_Correl_ID	Informe	
JMS_IBM_Report_Discard_Msg	Informe	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Comentarios	
JMS_IBM_Format	Formato	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding "1" en la página 169	Encoding	
JMS_IBM_Character_Set "1" en la página 169	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Sólo se establece si el mensaje entrante es un mensaje de bytes.

Intercambio de mensajes entre una aplicación de JMS y una aplicación de IBM MQ tradicional
Este tema describe qué sucede cuando una aplicación de JMS intercambia mensajes con una aplicación de IBM MQ tradicional que no puede procesar la cabecera MQRFH2.

La Figura 11 en la página 169 muestra la correlación.

El administrador indica que la aplicación de JMS se está comunicando con una aplicación de IBM MQ tradicional estableciendo la propiedad TARGCLIENT del destino en *MQ*. Esto indica que no se debe crear ninguna cabecera MQRFH2. Si no se realiza este paso, la aplicación receptora debe ser capaz de manejar la cabecera MQRFH2.

La correlación de JMS con MQMD dirigida a una aplicación IBM MQ tradicional es la misma que la correlación de JMS con MQMD dirigida a una aplicación JMS. Si IBM MQ classes for JMS recibe un mensaje IBM MQ con el campo de MQMD *Format* establecido en cualquier valor distinto a MQFMT_RFH2, los datos se están recibiendo de una aplicación no JMS. Si el formato es MQFMT_STRING, el mensaje se recibe como un mensaje de texto de JMS. En otro caso, se recibe como un mensaje de bytes de JMS. Debido a que no hay ninguna MQRFH2, sólo se pueden restaurar las propiedades de JMS que se transmiten en el MQMD.

Si IBM MQ classes for JMS recibe un mensaje que no tiene una cabecera MQRFH2, la propiedad TARGCLIENT del objeto Queue o Topic que deriva del campo de cabecera JMSReplyTo del mensaje se establece en MQ de forma predeterminada. Esto significa que un mensaje de respuesta que se envía a la cola o tema tampoco tiene una cabecera MQRFH2. Puede desactivar este comportamiento de incluir una cabecera MQRFH2 en un mensaje de respuesta solo si el mensaje original tiene una cabecera MQRFH2. Para ello establezca la propiedad TARGCLIENTMATCHING de la fábrica de conexiones en NO.

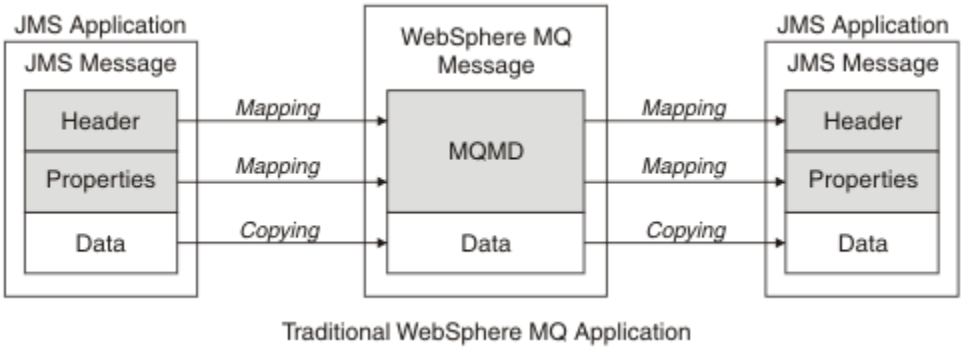


Figura 11. Cómo los mensajes de JMS se transforman en mensajes de IBM MQ sin ninguna cabecera MQRFH2

El cuerpo de mensaje de JMS
Este tema contiene información sobre la codificación del propio cuerpo del mensaje. La codificación utilizada depende del tipo de mensaje de JMS.

ObjectMessage
Un ObjectMessage es un objeto serializado por Java Runtime de la forma habitual.

TextMessage
TextMessage es una serie codificada. Para un mensaje saliente, la serie se codifica según el juego de caracteres proporcionado por el objeto de destino. Toma como valor predeterminado la codificación UTF8 (la codificación UTF8 empieza con el primer carácter del mensaje. No existe campo de longitud al principio). Pero es posible especificar cualquier otro juego de caracteres soportado por IBM MQ classes for JMS. Tales juegos de caracteres se utilizan principalmente cuando envía un mensaje a una aplicación que no es de JMS.

Si el juego de caracteres es un juego de doble byte (incluido UTF16), la codificación de enteros especificada por el objeto de destino determina el orden de los bytes.

Un mensaje entrante se interpreta utilizando el juego de caracteres y la codificación que están especificados en el propio mensaje. Estas especificaciones se encuentran en la última cabecera de IBM MQ (o MQMD si no hay cabeceras). Para los mensajes de JMS, la última cabecera es normalmente la MQRFH2.

BytesMessage

Un BytesMessage es, de forma predeterminada, una secuencia de bytes de acuerdo con lo definido por la especificación JMS 1.0.2 y la documentación asociada de Java.

Para un mensaje saliente que ha sido ensamblado por la propia aplicación, se puede utilizar la propiedad de codificación del objeto de destino para alterar temporalmente las codificaciones de los campos de coma flotante y entero contenidos en el mensaje. Por ejemplo, puede solicitar que los valores de coma flotante se almacenen en formato S/390, en lugar del formato IEEE.

Un mensaje entrante se interpreta utilizando la codificación numérica especificada en el mensaje. Esta especificación se encuentra en la última cabecera de IBM MQ (o en MQMD si no hay cabeceras). Para los mensajes de JMS, la última cabecera es normalmente la MQRFH2.

Si se recibe un BytesMessage y se vuelve a enviar sin realizar ninguna modificación, su cuerpo se transmite byte a byte, tal como se ha recibido. La propiedad de codificación del objeto de destino no tiene ningún efecto en el cuerpo. La única entidad similar a una serie que se pueden enviar explícitamente en un BytesMessage es una serie UTF8. Esto se codifica en formato UTF8 de Java, y se inicia con un campo de longitud de 2 bytes. La propiedad del juego de caracteres del objeto de destino no tiene ningún efecto en la codificación de un BytesMessage saliente. El valor de juego de caracteres en un mensaje entrante de IBM MQ no tiene ningún efecto en la interpretación de ese mensaje como BytesMessage de JMS.

Es poco probable que las aplicaciones que no son de Java reconozcan la codificación UTF8 de Java. Por lo tanto, para que una aplicación de JMS envíe un BytesMessage que contiene datos de texto, la propia aplicación debe convertir sus series en matrices de bytes y escribir estas matrices de bytes en el BytesMessage.

MapMessage

MapMessage es una serie que contiene los tríos nombre/tipo/valor XML codificados como:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

donde `datatype` es uno de los tipos de datos listados en Tabla 21 en la página 159. El tipo de datos predeterminado es `string`, por lo que el atributo `dt="string"` se omite para los elementos de tipo serie.

El juego de caracteres que se utiliza para codificar o interpretar la serie XML que forma el cuerpo de un mensaje de correlación se determina de acuerdo con las reglas que se aplican a un mensaje de texto.

Las versiones de IBM MQ classes for JMS anteriores a 5.3 codificaban el cuerpo de un mensaje de correlación en el formato siguiente:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 y posteriores pueden interpretar cualquier formato, pero las versiones de IBM MQ classes for JMS anteriores a 5.3 no pueden interpretar el formato actual.

Si una aplicación necesita enviar mensajes de correlación a otra aplicación que esté utilizando una versión de IBM MQ classes for JMS anterior a 5.3, la aplicación emisora debe llamar al método de fábrica de conexiones `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)`

para especificar que los mensajes de correlación se envíen en el formato anterior. De forma predeterminada, todos los mensajes de correlación se envían con el formato actual.

StreamMessage
Un StreamMessage es como un mensaje de correlación, pero sin nombres de elemento:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

donde `datatype` es uno de los tipos de datos listados en Tabla 21 en la página 159. El tipo de datos predeterminado es `string`, por lo que el atributo `dt="string"` se omite para los elementos de tipo serie.

El juego de caracteres que se utiliza para codificar o interpretar la serie XML que integra el cuerpo del StreamMessage se determina siguiendo las normas que se aplican a un TextMessage.

El campo MQRFH2.format se establece tal como se indica a continuación:

MQFMT_NONE
para ObjectMessage, BytesMessage o mensajes sin cuerpo.

MQFMT_STRING
para TextMessage, StreamMessage o MapMessage.

Conversión de mensajes JMS
La conversión de datos de mensajes en JMS se realiza cuando se envían y reciben mensajes. IBM MQ realiza la mayor parte de la conversión de datos automáticamente. Convierte datos de texto y datos numéricos al transferir un mensaje entre aplicaciones JMS. El texto se convierte cuando se intercambia un `JMSTextMessage` entre una aplicación JMS y una aplicación IBM MQ.

Si tiene pensado realizar intercambios más complejos de mensajes, le interesarán los temas siguientes. Entre los intercambios complejos de mensajes se incluyen:

- Transferencia de mensajes no de texto entre una aplicación IBM MQ y una aplicación JMS.
- Intercambio de datos de texto en formato de byte.
- Conversión de texto en la aplicación.

Datos de mensaje JMS

La conversión de datos es necesaria para intercambiar datos de texto y numéricos entre aplicaciones, incluso entre dos aplicaciones JMS. La representación interna del texto y los números tiene que estar codificada para que puedan ser transferidos en un mensaje. La codificación obliga a decidir de qué forma se representan los números y el texto. IBM MQ gestiona la codificación de texto y números en mensajes JMS, excepto para `JMSObjectMessage`, consulte "JMSObjectMessage" en la página 178. Utiliza tres atributos de mensaje. Estos son `CodedCharacterSetId`, `Encoding` y `Format`.

Estos tres atributos de mensaje se suelen almacenar en la cabecera JMS, `MQRFH2`, los campos de un mensaje JMS. Si el tipo de mensaje es `MQ`, en lugar de `JMS`, los atributos se almacenan en el descriptor de mensaje, `MQMD`. Los atributos se utilizan para convertir los datos de mensaje JMS. Los datos de mensaje JMS se transfieren en la parte de datos de mensaje de un mensaje IBM MQ.

Propiedades de mensaje JMS

Las propiedades de mensaje JMS, como `JMS_IBM_CHARACTER_SET`, se intercambian en la parte de cabecera `MQRFH2` de un mensaje JMS, a menos que el mensaje se haya enviado sin un `MQRFH2`. Solo `JMSTextMessage` y `JMSBytesMessage` pueden enviarse sin una `MQRFH2`. Si una propiedad JMS se almacena como una propiedad de mensaje IBM MQ en el descriptor del mensaje, `MQMD`, se convierte como parte de la conversión de `MQMD`. Si una propiedad JMS se almacena en `MQRFH2`, se almacena en el juego de caracteres especificado por `MQRFH2.NameValueCCSID`. Cuando un mensaje se envía o recibe,

sus propiedades se convierten a y desde su representación interna en la JVM. La conversión se hace hacia y desde el juego de caracteres del descriptor de mensaje o `MQRFH2.NameValueCCSID`. Los datos numéricos se convierten en texto.

Conversión de mensajes JMS

Los temas siguientes contienen ejemplos y tareas que son útiles si tiene previsto intercambiar mensajes más complejos que requieran una conversión.

Enfoques de conversión de mensajes JMS
Hay una serie de enfoques de conversión de datos disponibles para los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si su aplicación solo intercambia texto o intercambia mensajes únicamente con otras aplicaciones JMS, normalmente no se plantearía la conversión de datos. La conversión de datos se realiza automáticamente, mediante IBM MQ.

Puede realizar una serie de preguntas acerca de cómo enfocar la conversión de mensajes:

¿Es realmente necesario pensar en la conversión de mensajes?
En algunos casos, como en las transferencias de mensajes de JMS a JMS, y en el intercambio de mensajes de texto con programas IBM MQ, IBM MQ realiza las conversiones necesarias automáticamente. Es posible que desee controlar la conversión de datos por motivos de rendimiento, o puede que intercambie mensajes complejos que tienen un formato predefinido. En casos como estos, debe entender la conversión de mensajes y leer los temas siguientes.

¿Qué tipos de conversión existen?
Hay cuatro tipos principales de conversión, que se explican en las secciones siguientes:

1. “Conversión de datos de cliente JMS” en la página 172
2. “Conversión de datos de aplicación” en la página 173
3. “Conversión de datos de gestor de colas” en la página 174
4. “Conversión de datos de canal de mensajes” en la página 174

¿Dónde se debe realizar la conversión?
En la sección “Elección de un enfoque para la conversión de mensajes: el receptor lo hace bien” en la página 174 se describe el enfoque habitual de "el receptor lo hace bien". "El receptor lo hace bien" también se aplica a la conversión de datos JMS.

Conversión de datos de cliente JMS

JMS cliente[1]La conversión de datos es la conversión de primitivas y objetos Java en bytes en un mensaje JMS cuando se envía a un destino, y la conversión inversa cuando se recibe. La conversión de datos de cliente JMS utiliza los métodos de las clases `JMSMessage`. Los métodos los muestra el tipo de clase `JMSMessage` en Tabla 31 en la página 176.

La conversión a y desde la representación JVM interna de números y texto se realiza para los métodos de lectura, obtención, establecimiento y escritura. Se realiza la conversión cuando se envía un mensaje, y cuando se llama a cualquier de los métodos de lectura u obtención en un mensaje recibido.

La página de códigos y la codificación numérica utilizadas para escribir o establecer el contenido de un mensaje se definen como atributos del destino. La página de códigos y la codificación numérica del destino se pueden cambiar de forma administrativa. Una aplicación puede también alterar temporalmente la página de códigos y la codificación de destino estableciendo propiedades de mensaje que controlan la escritura o establecimiento del contenido del mensaje.

Si desea convertir la codificación numérica cuando se envía un mensaje `JMSBytesMessage` a un destino que no está definido como codificación `Native`, debe establecer la propiedad de mensaje `JMS_IBM_ENCODING` antes de enviar el mensaje. Si sigue el patrón de "el receptor lo hace bien",

[1] "JMS Cliente" hace referencia al IBM MQ classes for JMS que implementa la interfaz JMS , que se ejecuta en modalidad de cliente o de enlaces.

o si intercambia mensajes entre aplicaciones JMS, no es necesario que la aplicación establezca JMS_IBM_ENCODING. En la mayoría de los casos, puede dejar la propiedad Encoding como Native.

Para los mensajes JMSStreamMessage, JMSMapMessage y JMSTextMessage, se utilizan las propiedades del identificador de juego de caracteres del destino. La codificación se pasa por alto en el envío, ya que los números se escriben en formato de texto. El programa de aplicación cliente JMS no tiene que establecer JMS_IBM_CHARACTER_SET antes de enviar el mensaje si se va a aplicar la propiedad del juego de caracteres de destino.

Para obtener los datos de un mensaje, un aplicación llama a los métodos de lectura u obtención de mensajes JMS. Los métodos hacen referencia a la página de códigos y la codificación definidas en la cabecera del mensaje anterior para crear las primitivas y objetos Java correctamente.

La conversión de datos de cliente JMS cumple las necesidades de la mayoría de las aplicaciones JMS que intercambian mensajes entre un cliente JMS y otro. No codifique ninguna conversión de datos explícita. No utilice la clase java.nio.charset.Charset, que normalmente se utiliza al escribir texto en un archivo. Los métodos writeString y setString realizan la conversión automáticamente.

Para obtener más detalles sobre la conversión de datos de cliente JMS, consulte "Conversión y codificación de mensajes de cliente JMS" en la página 185.

Conversión de datos de aplicación

Una aplicación cliente JMS puede realizar una conversión de datos de caracteres explícita utilizando la clase java.nio.charset.Charset; consulte los ejemplos de Figura 14 en la página 177 y Figura 15 en la página 177. Los datos de serie se convierten en bytes, utilizando el método getBytes, y se envían como bytes. Los bytes vuelven a convertirse en texto utilizando el constructor de String, que acepta una matriz de bytes y un Charset. Los datos de caracteres se convierten utilizando los métodos de Charset encode y decode. Normalmente, el mensaje se envía o recibe como JMSBytesMessage, porque la parte del mensaje de un JMSBytesMessage no contiene nada que no sean los datos grabados por la aplicación[2]. También puede enviar y recibir bytes utilizando JMSStreamMessage, JMSMapMessageo JMSObjectMessage.

No existen métodos Java para codificar y decodificar bytes que contienen datos numéricos representados en distintos formatos de codificación. Los datos numéricos se codifican y decodifican automáticamente utilizando los métodos de lectura y escritura numéricos de JMSMessage. Los métodos de lectura y escritura utilizan el valor del atributo JMS_IBM_ENCODING de los datos de mensaje.

Un uso típico de la conversión de datos de aplicación se produce cuando un cliente JMS envía o recibe un mensaje formateado desde una aplicación que no es JMS. Un mensaje formateado contiene datos de texto, numéricos y bytes organizados por la longitud de los campos de datos. A menos que la aplicación no JMS haya especificado el formato de mensaje como "MQSTR ", el mensaje se construirá como un JMSBytesMessage. Para recibir los datos del mensaje formateado en un JMSBytesMessage, debe llamar a una secuencia de métodos. Los métodos se deben llamar en el mismo orden en que los campos se escribieron en el mensaje. Si los campos son numéricos, debe conocer la codificación y la longitud de los datos numéricos. Si alguno de los campos contiene datos de bytes o texto, debe conocer la longitud de los datos de bytes del mensaje. Hay dos maneras de convertir un mensaje formateado en un objeto Java que sea fácil de utilizar.

1. Construir una clase Java correspondiente al registro para encapsular la lectura y la escritura del mensaje. El acceso a los datos del registro se lleva a cabo con los métodos de obtención (get) y establecimiento (set) de la clase.
2. Construir una clase Java correspondiente al registro extendiendo la clase com.ibm.mq.headers. El acceso a los datos de la clase es con accesores específicos de tipo con el formato getStringValue(*fieldname*);

Consulte "Intercambio de un registro formateado con una aplicación no JMS" en la página 193.

[2] Una excepción: los datos escritos utilizando writeUTF empiezan con un campo de longitud de 2 bytes

Conversión de datos de gestor de colas

El gestor de colas puede realizar la conversión de páginas de códigos cuando un programa cliente JMS obtiene un mensaje. La conversión es la misma que la conversión que se realiza para un programa C. Un programa C establece MQGMO_CONVERT como una opción de parámetro MQGET GetMsgOpts ; consulte Figura 13 en la página 177. Un gestor de colas realiza la conversión para un programa cliente JMS que recibe un mensaje, si la propiedad de destino WMQ_RECEIVE_CONVERSION está establecida en WMQ_RECEIVE_CONVERSION_QMGR. El programa cliente JMS también puede establecer la propiedad de destino; consulte Figura 12 en la página 174.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

O,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 12. Habilitar la conversión de datos de gestor de colas

La ventaja principal de la conversión de gestor de colas queda de manifiesto al intercambiar mensajes con aplicaciones que no son JMS. Si el campo Format del mensaje está definido y el juego de caracteres de destino, o la codificación, es diferente al del mensaje, el gestor de colas realiza la conversión de datos para la aplicación de destino, si la aplicación lo solicita. El gestor de colas convierte los datos de mensaje formateados de acuerdo con uno de los tipos de mensaje IBM MQ predefinidos, como una cabecera CICS bridge (MQCIH). Si el campo Format está definido por el usuario, el gestor de colas busca una salida de conversión de datos con el nombre proporcionado en el campo Format .

La conversión de datos de gestor de colas se utiliza con el patrón de diseño de "el receptor lo hace bien" para sacar su máximo partido. Un cliente JMS de envío no necesita realizar la conversión. Un programa de recepción que no sea JMS se basa en la salida de conversión para garantizar que se entrega el mensaje en la página de códigos y la codificación necesarias. Con un cliente JMS emisor y un receptor noJMS , el ejemplo se aplica a IBM MQ.

Puede crear una salida de conversión de datos, utilizando el programa de utilidad de salida de conversión de datos, **crtmqcvx**, para permitir que el gestor de colas pueda convertir sus propios datos con formato de registro. Puede crear su propio formato de registro, utilizar com.ibm.mq.headers para acceder a él como una clase Java y utilizar su propias salida de conversión para convertirlo. En z/OS, el programa de utilidad se denomina **CSQUCVX**, y en IBM i, **CVTMQMDTA**. Consulte “Intercambio de un registro formateado con una aplicación no JMS” en la página 193

Conversión de datos de canal de mensajes

Los canales Emisor, Servidor, Clúster receptor y Clúster emisor de IBM MQ tienen una opción de conversión de mensajes, CONVERT. El contenido de un mensaje se puede convertir de forma opcional cuando se envía un mensaje. La conversión se lleva a cabo en el extremo de envío del canal. La definición de clúster receptor se utiliza para definir automáticamente el canal de clúster emisor correspondiente.

La conversión de datos por canales de mensajes se utiliza normalmente si no es posible utilizar otras formas de conversión.

Elección de un enfoque para la conversión de mensajes: "el receptor lo hace bien"

El enfoque habitual en el diseño de aplicaciones IBM MQ para la conversión de código es "el receptor lo hace bien". "El receptor lo hace bien" reduce el número de conversiones de mensajes. También evita el problema de que se produzcan errores de canal no esperados si falla la conversión de mensajes en algún gestor de colas intermedio durante la transferencia de mensajes. La regla "el receptor lo hace bien" solo

se rompe si hay algún motivo por el que el receptor no lo puede hacer bien. La plataforma de recepción puede no tener el juego de caracteres correcto, por ejemplo.

"El receptor lo hace bien" también es una buena guía general para las aplicaciones cliente JMS. Pero, en casos específicos, la conversión al juego de caracteres correcto en el origen puede ser más eficiente. La conversión de la representación interna JVM debe llevarse a cabo cuando se envía un mensaje que contiene tipos numéricos o de texto. La conversión al juego de caracteres necesario para el receptor, si el receptor no es un cliente JMS, podría eliminar la necesidad de que el destinatario no JMS realice la conversión. Si el destinatario es un cliente JMS, realizará la conversión de nuevo de todos modos, para decodificar los datos de mensaje y crear primitivas y objetos Java.

La diferencia entre las aplicaciones cliente JMS y aplicaciones escritas en un lenguaje como C, es que Java debe realizar una conversión de datos. Una aplicación Java debe convertir números y texto de su representación interna a un formato codificado utilizado en los mensajes.

Al establecer el destino o las propiedades de mensaje, puede establecer el juego de caracteres y la codificación que utiliza IBM MQ para codificar números y texto en los mensajes. Normalmente, dejaría el juego de caracteres como `1208` y la codificación como `Native`.

IBM MQ no convierte matrices de bytes. Para codificar series y matrices de caracteres en matrices de bytes, utilice el paquete `java.nio.charset`. `Charset` especifica el juego de caracteres utilizado para convertir una serie o matriz de caracteres en una matriz de bytes. También puede decodificar una matriz de bytes en una serie o matriz de caracteres utilizando un `Charset`. No es una buena práctica basarse en `java.nio.charset.Charset.defaultCodePage` al codificar series y matrices de caracteres. El `Charset` predeterminado es normalmente `windows-1252` en Windows, y `UTF-8` en AIX and Linux. `windows-1252` es un juego de caracteres de un solo byte y `UTF-8` es un juego de caracteres de varios bytes.

Normalmente, debe dejar las propiedades de juego de caracteres y codificación de destino en sus valores predeterminados de `UTF-8` y `Native` al intercambiar mensajes con otras aplicaciones JMS. Si intercambia mensajes que contienen números o texto con una aplicación JMS, elija el tipo de mensaje `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage` o `JMSObjectMessage` que se ajuste a sus necesidades. No hay ninguna otra tarea de conversión que hacer.

Si está intercambiando mensajes con aplicaciones no JMS que utilizan un formato de registro, es más complicado. A menos que el registro completo contenga texto y se pueda transferir como un `JMSTextMessage`, debe codificar y decodificar el texto en la aplicación. Establezca el tipo de mensaje de destino en `MQ` y utilice `JMSBytesMessage` para evitar que IBM MQ classes for JMS añada información de etiquetado y cabecera adicional a los datos del mensaje. Utilice los métodos `JMSBytesMessage` para escribir números y bytes, y la clase `Charset` para convertir el texto en matrices de bytes explícitamente. Hay una serie de factores que pueden influir en su elección del juego de caracteres:

- Rendimiento: ¿puede reducir el número de conversiones transformando el texto en un juego de caracteres que se utilice en el mayor número de servidores?
- Uniformidad: transfiera todos los mensajes en el mismo juego de caracteres.
- Riqueza: ¿qué juegos de caracteres tienen todos los puntos de código que deben utilizar las aplicaciones?
- Simplicidad: los juegos de caracteres de un solo byte son más fáciles de utilizar que los juegos de caracteres de varios bytes y de longitud variable.

Consulte “Intercambio de un registro formateado con una aplicación no JMS” en la página 193. para ver ejemplos de cómo convertir mensajes intercambiados con aplicaciones no JMS.

Ejemplos

Tabla de tipos de mensajes y tipos de conversión

Tabla 31. Tipos de mensajes y tipos de conversión

	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
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

```
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                     */
```

Figura 13. Fragmento de código de amqsget0.c

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 14 en la página 177 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir una serie de texto en un mensaje de bytes que contiene una combinación de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 15 en la página 177. Incluso para una serie con un número fijo de caracteres, la longitud de la representación de bytes podría ser mayor.

```
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);
```

Figura 14. Envío de una String en un JMSBytesMessage

```
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);
```

Figura 15. Recepción de una String de un JMSBytesMessage

Conceptos relacionados

Conversión y codificación de mensajes de cliente JMS
Se muestran los métodos que puede utilizar para realizar la conversión y codificación de mensajes de cliente JMS, con ejemplos de código de cada tipo de conversión.

Conversión de datos de gestor de colas
La conversión de datos de gestor de colas siempre ha estado disponible para las aplicaciones que no son JMS y que reciben mensajes de clientes JMS. Los clientes de JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas, que es opcional.

Tareas relacionadas

Intercambio de un registro formateado con una aplicación no JMS
Siga los pasos sugeridos en esta tarea para diseñar y crear una salida de conversión de datos, y una aplicación cliente JMS que pueda intercambiar mensajes con una aplicación no JMS utilizando JMSBytesMessage. El intercambio de un mensaje con formato con una aplicación no JMS se puede producir con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensaje JMS
La elección del tipo de mensaje afecta a la manera en que se enfoca la conversión de mensajes. La interacción de la conversión de mensajes y el tipo de mensaje se describe para los tipos de mensaje JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage y JMSBytesMessage.

Conversión y tipos de mensaje JMS
La elección del tipo de mensaje afecta a la manera en que se enfoca la conversión de mensajes. La interacción de la conversión de mensajes y el tipo de mensaje se describe para los tipos de mensaje JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage y JMSBytesMessage.

JMSObjectMessage

JMSObjectMessage contiene un objeto y cualquier objeto que referencie, serializado en una secuencia de bytes por la JVM. El texto se serializa en UTF-8 y se limita a cadenas o vectores de caracteres de no más de 65534 bytes. Una ventaja de JMSObjectMessage es que las aplicaciones no se implican en ningún problema de conversión de datos siempre que se limiten a usar los métodos y atributos del objeto. JMSObjectMessage proporciona la conversión de datos de objetos complejos sin que el programador de aplicaciones se tenga que plantear cómo codificar un objeto en un mensaje. La desventaja de utilizar JMSObjectMessage es que solo se puede intercambiar con otras aplicaciones JMS. Si se elige uno de los otros tipos de mensaje JMS, es posible intercambiar mensajes JMS con aplicaciones no JMS.

"Envío y recepción de un JMSObjectMessage" en la página 181 muestra un objeto String que se intercambia en un mensaje.

Una aplicación cliente JMS puede recibir un JMSObjectMessage solo en un mensaje que tenga un cuerpo de estilo JMS. El destino debe especificar un cuerpo de estilo JMS.

JMSTextMessage

JMSTextMessage contiene una única cadena de texto. Cuando se envía un mensaje de texto, el Format del texto se establece en "MQSTR ", WMQConstants.MQFMT_STRING. El CodedCharacterSetId del texto se establece al identificador de juego de caracteres codificados del destino. El texto se codifica en CodedCharacterSetId mediante IBM MQ. Los campos CodedCharacterSetId y Format se establecen en el descriptor de mensaje, MQMD, o en los campos de JMS en MQRFH2. Si el mensaje se ha definido para que tenga un estilo de cuerpo de mensaje WMQ_MESSAGE_BODY_MQ, o dicho estilo de cuerpo no se ha especificado, pero el destino objetivo es WMQ_TARGET_DEST_MQ, se configuran los campos descriptores de mensaje. De lo contrario, el mensaje tiene un JMS RFH2 y los campos se establecen en la parte fija de MQRFH2.

Una aplicación puede sustituir el identificador de juego de caracteres codificados definido para un destino. Tiene que establecer la propiedad de mensaje JMS_IBM_CHARACTER_SET a un identificador de juego de caracteres codificados; consulte el ejemplo en "Envío y recepción de un JMSTextmessage" en la página 181.

Cuando el cliente JMS llama al método consumer.receive, la conversión del gestor de colas es opcional. La conversión del gestor de colas se habilita estableciendo la propiedad de destino WMQ_RECEIVE_CONVERSION a WMQ_RECEIVE_CONVERSION_QMGR. El gestor de colas convierte el mensaje de texto del JMS_IBM_CHARACTER_SET especificado para el mensaje antes de transferir el mensaje al cliente JMS. El juego de caracteres del mensaje convertido es 1208, UTF-8, a menos que el destino tenga un WMQ_RECEIVE_CCSID distinto. El CodedCharacterSetId en el mensaje que hace referencia al JMSTextMessage se actualiza al ID de juego de caracteres de destino. El texto se

descodifica del juego de caracteres de destino a Unicode con el método `getText`; consulte el ejemplo en "Envío y recepción de un JMSTextmessage" en la página 181.

Se puede enviar un `JMSTextMessage` en un cuerpo de mensaje de estilo MQ, sin una cabecera JMS `MQRFH2`. El valor de los atributos de destino, `WMQ_MESSAGE_BODY` y `WMQ_TARGET_DEST` determinan el estilo de cuerpo del mensaje, a menos que la aplicación los sustituya. La aplicación puede sustituir los valores configurados en el destino invocando `destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ)` o `destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)`.

Si se envía un `JMSTextMessage` con un cuerpo de estilo MQ enviándolo a un destino con `WMQ_MESSAGE_BODY` establecida a `WMQ_MESSAGE_BODY_MQ`, no se podrá recibir como un `JMSTextMessage` procedente de ese mismo destino. Todos los mensajes recibidos de un destino con `WMQ_MESSAGE_BODY` establecida a `WMQ_MESSAGE_BODY_MQ` se reciben como un `JMSBytesMessage`. Si intenta recibir el mensaje como `JMSTextMessage` , se genera una excepción, `ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage`.

Nota: El cliente JMS no convierte el texto de un `JMSBytesMessage`. El cliente solo puede recibir el texto del mensaje como un vector de bytes. Si la conversión de gestor de colas está habilitada, este convertirá el texto, pero el cliente JMS aún tendrá que recibirlo como un vector de bytes en un `JMSBytesMessage`.

Por lo general, es mejor usar la propiedad `WMQ_TARGET_DEST` para controlar si un `JMSTextMessage` se envía con un estilo de cuerpo MQ o JMS. Luego se podrá recibir el mensaje de un destino que tenga `WMQ_TARGET_DEST` establecida a `WMQ_TARGET_DEST_MQ` o `WMQ_TARGET_DEST_JMS`. `WMQ_TARGET_DEST` no tiene ningún efecto en el receptor.

JMSMapMessage y JMSStreamMessage

Estos dos tipos de mensaje JMS son similares. Se pueden leer y escribir tipos primitivos en los mensajes utilizando los métodos basados en las interfaces `DataInputStream` y `DataOutputStream`; consulte "Tabla de tipos de mensajes y tipos de conversión" en la página 183. Los detalles se describen en "Conversión y codificación de mensajes de cliente JMS" en la página 185. Cada primitiva está etiquetada; consulte "El cuerpo de mensaje de JMS" en la página 169.

Los datos numéricos se leen y se escriben en el mensaje codificados como texto XML. No se hace ninguna referencia a la propiedad de destino, `JMS_IBM_ENCODING`. Los datos de texto reciben el mismo tratamiento que el texto de un `JMSTextMessage`. Si se mirase el contenido del mensaje creado en el ejemplo Figura 20 en la página 182, todos los datos del mensaje estarían en EBCDIC, como si se hubieran enviado con el juego de caracteres 37.

Se pueden enviar varios elementos en un `JMSMapMessage` o en un `JMSStreamMessage`.

Los elementos individuales se pueden recuperar por nombre en un `JMSMapMessage`, o por posición en un `JMSStreamMessage`. Cada elemento se decodifica cuando se invoca un método get o read utilizando el valor `CodedCharacterSetId` almacenado en el mensaje. Si el método utilizado para recuperar el elemento devuelve un tipo diferente del tipo enviado, este se convertirá. Si no se puede convertir el tipo, se generará una excepción. Consulte Clase JMSStreamMessage para obtener detalles. El ejemplo "Envío de datos en un JMSStreamMessage y en un JMSMapMessage" en la página 182 ilustra la conversión de tipos y la obtención del contenido de un `JMSMapMessage` fuera de secuencia.

El campo `MQRFH2.format` para `JMSMapMessage` y `JMSStreamMessage` se establece en "MQSTR ". Si la propiedad de destino `WMQ_RECEIVE_CONVERSION` se establece en `WMQ_RECEIVE_CONVERSION_QMGR`, los datos del mensaje se convierten mediante el gestor de colas antes de enviarse al cliente de JMS. El `MQRFH2.CodedCharacterSetId` del mensaje es el `WMQ_RECEIVE_CCSID` del destino. El `MQRFH2.Encoding` es `Native`. Si `WMQ_RECEIVE_CONVERSION` es `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`, `CodedCharacterSetId` y `Encoding` de `MQRFH2` tendrán el valor estbalecido por el emisor.

Una aplicación cliente de JMS puede recibir un `JMSMapMessage` o un `JMSStreamMessage` solo en un mensaje que tenga un cuerpo de estilo JMS y de un destino que no especifique un cuerpo de estilo MQ.

JMSBytesMessage

Un JMSBytesMessage puede contener varios tipos primitivos. Se pueden leer y escribir tipos primitivos en los mensajes utilizando los métodos basados en las interfaces DataInputStream y DataOutputStream; consulte "Tabla de tipos de mensajes y tipos de conversión" en la página 183. Los detalles se describen en "Conversión y tipos de mensaje JMS" en la página 178.

La codificación de datos numéricos del mensaje se controla con el valor de JMS_IBM_ENCODING que se establece antes de escribir dichos datos en el JMSBytesMessage. Una aplicación puede sustituir la codificación predeterminada Native definida para JMSBytesMessage configurando la propiedad de mensaje JMS_IBM_ENCODING.

Los datos de texto se pueden leer y escribir en UTF-8 utilizando los métodos readUTF y writeUTF, o bien en Unicode utilizando los métodos readChar y writeChar . No hay ningún método que utilice CodedCharacterSetId. De forma alternativa, el cliente JMS puede codificar y descodificar el texto en bytes utilizando la clase Charset. Transfiere los bytes entre la JVM y el mensaje sin que las IBM MQ classes for JMS realicen ninguna conversión; consulte "Envío y recepción de texto en un JMSBytesMessage" en la página 182.

Normalmente, se envía un JMSBytesMessage enviado a una aplicación MQ en un cuerpo de mensaje de estilo MQ, sin una cabecera JMS MQRFH2. Si se envía a una aplicación JMS, el estilo del cuerpo del mensaje suele ser JMS. El valor de los atributos de destino, WMQ_MESSAGE_BODY y WMQ_TARGET_DEST determinan el estilo de cuerpo del mensaje, a menos que la aplicación los sustituya. La aplicación puede sustituir los valores configurados en el destino invocando destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) o destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Si se envía un JMSBytesMessage con un cuerpo de estilo MQ, se puede recibir el mensaje de un destino que defina un estilo de cuerpo de mensaje MQ o JMS. Si se envía un JMSBytesMessage con un cuerpo de estilo JMS, habrá que recibir el mensaje de un destino que defina un estilo de cuerpo del mensaje JMS. En caso contrario, MQRFH2 se tratará como parte de los datos del mensaje del usuario, lo que podría no ser el comportamiento deseado.

Tanto si un mensaje tiene un estilo de cuerpo MQ como JMS, la forma en que se recibe no se ve afectada por la configuración de WMQ_TARGET_DEST.

El gestor de colas podría transformar el mensaje posteriormente si se proporciona un Format de los datos del mensaje y la conversión de datos de gestor de colas está habilitada. No utilice el campo de formato para nada más que especificar el formato de los datos del mensaje, o déjelo en blanco, MQConstants.MQFMT_NONE

Se pueden enviar varios elementos en un JMSBytesMessage. Se convertirá cada elemento numérico cuando se envíe el mensaje utilizando la codificación definida para el mensaje.

Los elementos individuales de los datos se pueden recuperar de JMSBytesMessage. Invoque los métodos de lectura en el mismo orden en que se invocaron los métodos de escritura para crear el mensaje. Cuando se invoca el mensaje, cada elemento numérico se convierte utilizando el valor de Encoding almacenado en el mensaje.

A diferencia de JMSMapMessage y JMSStreamMessage, JMSBytesMessage solo contiene datos escritos por la aplicación. En los datos del mensaje no se almacenan datos adicionales como, por ejemplo, las etiquetas XML usadas para definir los elementos de un JMSMapMessage y de un JMSStreamMessage. Por este motivo, utilice JMSBytesMessage para transferir mensajes formateados para otras aplicaciones.

La conversión entre JMSBytesMessage y DataInputStream y DataOutputStream es útil en algunas aplicaciones. Se necesitará un código basado en el ejemplo "Lectura y escritura de mensajes con DataInputStream y DataOutputStream" en la página 182 para usar el paquete com.ibm.mq.header con JMS.

Ejemplos

Envío y recepción de un JMSObjectMessage

```
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
```

Figura 16. Envío y recepción de un JMSObjectMessage

Envío y recepción de un JMSTextmessage

Un mensaje de texto no puede contener texto en juegos de caracteres distintos. El ejemplo muestra texto en juegos de caracteres distintos, enviados en dos mensajes diferentes.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 17. Enviar texto de mensaje en el juego de caracteres definido por el destino

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 18. Enviar mensaje de texto en ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 19. Recibir mensaje de texto

Envío de datos en un JMSStreamMessage y en un JMSMapMessage

```
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
```

Figura 20. Envío de datos en un JMSStreamMessage y en un JMSMapMessage

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 21 en la página 182 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir una serie de texto en un mensaje de bytes que contiene una combinación de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 22 en la página 182. Incluso para una serie con un número fijo de caracteres, la longitud de la representación de bytes podría ser mayor.

```
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);
```

Figura 21. Envío de una String en un JMSBytesMessage

```
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);
```

Figura 22. Recepción de una String de un JMSBytesMessage

Lectura y escritura de mensajes con DataInputStream y DataOutputStream

El código en Figura 23 en la página 183 crea un JMSBytesMessage usando un DataOutputStream.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

Figura 23. Envío de un JMSBytesMessage con un DataOutputStream

La sentencia que establece la propiedad JMS_IBM_ENCODING está comentada. La sentencia es válida, si se escribe directamente en un JMSBytesMessage, pero no tiene ningún efecto cuando se escribe en DataOutputStream. Los números que se escriben en un DataOutputStream están codificados en Native. La configuración de JMS_IBM_ENCODING no tiene ningún efecto.

El código en Figura 24 en la página 183 recibe un JMSBytesMessage usando un DataInputStream.

```
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)));
```

Figura 24. Recepción de un JMSBytesMessage con un DataInputStream

La página de códigos se imprime utilizando la propiedad de página de códigos de los datos del mensaje de entrada, JMS_IBM_CHARACTER_SET. En la entrada, JMS_IBM_CHARACTER_SET es una página de códigos Java y no un identificador de juego de caracteres codificados numérico.

Tabla de tipos de mensajes y tipos de conversión

Tabla 32. Tipos de mensajes y tipos de conversión

	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tabla 32. Tipos de mensajes y tipos de conversión (continuación)				
	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
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

Conceptos relacionados

Enfoques de conversión de mensajes JMS

Hay una serie de enfoques de conversión de datos disponibles para los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si su aplicación solo intercambia texto o intercambia mensajes únicamente con otras aplicaciones JMS, normalmente no se plantearía la conversión de datos. La conversión de datos se realiza automáticamente, mediante IBM MQ.

Conversión y codificación de mensajes de cliente JMS

Se muestran los métodos que puede utilizar para realizar la conversión y codificación de mensajes de cliente JMS, con ejemplos de código de cada tipo de conversión.

Conversión de datos de gestor de colas

La conversión de datos de gestor de colas siempre ha estado disponible para las aplicaciones que no son JMS y que reciben mensajes de clientes JMS. Los clientes de JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas, que es opcional.

Tareas relacionadas

Intercambio de un registro formateado con una aplicación no JMS
Siga los pasos sugeridos en esta tarea para diseñar y crear una salida de conversión de datos, y una aplicación cliente JMS que pueda intercambiar mensajes con una aplicación no JMS utilizando `JMSBytesMessage`. El intercambio de un mensaje con formato con una aplicación no JMS se puede producir con o sin llamar a una salida de conversión de datos.

Conversión y codificación de mensajes de cliente JMS
Se muestran los métodos que puede utilizar para realizar la conversión y codificación de mensajes de cliente JMS, con ejemplos de código de cada tipo de conversión.

La conversión y la codificación se producen cuando se leen o escriben objetos o primitivas Java en los mensajes JMS. La conversión se denomina conversión de datos de cliente JMS para distinguirla de la conversión de datos de gestor de colas y la conversión de datos de aplicación. La conversión tiene lugar estrictamente cuando se leen o se escriben los datos en un mensaje JMS. El texto se convierte a y desde la representación Unicode interna de 16 bits[3]para el juego de caracteres utilizado para el texto de los mensajes. Los datos numéricos se convierten de los tipos numéricos primitivos de Java a la codificación definida para el mensaje. Si la conversión se realiza o no, y qué tipo de conversión se realiza, dependerá del tipo de mensaje JMS y de la operación de lectura o escritura.

Tabla 33 en la página 185 categoriza los métodos de lectura y escritura para distintos tipos de mensajes JMS según el tipo de conversión que se realiza. Los tipos de conversión se describen en el texto que sigue a la tabla.

Tabla 33. Tipos de mensajes y tipos de conversión

	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
`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`

[3] Alguna representación Unicode requiere más de 16 bits. Consulte una Referencia de Java SE

Tabla 33. Tipos de mensajes y tipos de conversión (continuación)				
	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
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

Texto

El CodedCharacterSetId predeterminado para un destino es 1208, UTF-8. De forma predeterminada, el texto se convierte de Unicode y se envía como una serie de texto UTF-8. En la recepción, el texto se convierte del juego de caracteres codificado en el mensaje recibido por el cliente a Unicode.

Los métodos setText y writeString convierten texto de Unicode al juego de caracteres definido para el destino. Una aplicación puede alterar temporalmente el juego de caracteres de destino estableciendo la propiedad de mensaje JMS_IBM_CHARACTER_SET. JMS_IBM_CHARác_SET, cuando se envía un mensaje debe ser un identificador de juego de caracteres codificado numérico[4].

Los fragmentos de código de "Envío y recepción de un JMSTextmessage" en la página 189 envían dos mensajes. Uno se envía en el juego de caracteres definido para el destino y el otro en el juego de caracteres 37, definido por la aplicación.

Los métodos getText y readString convierten el texto del mensaje del juego de caracteres definido en el mensaje a Unicode. Los métodos utilizan la página de códigos definida en la propiedad de mensaje JMS_IBM_CHARACTER_SET. La página de códigos se correlaciona a partir de MQRFH2.CodedCharacterSetId, a menos que el mensaje sea un mensaje de tipo MQ y no tenga MQRFH2. Si el mensaje es un mensaje de tipo MQ, sin MQRFH2, la página de códigos se correlaciona a partir de MQMD.CodedCharacterSetId.

El fragmento de código de Figura 29 en la página 189 recibe el mensaje que se ha enviado al destino. El texto del mensaje se convierte de la página de códigos IBM037 a Unicode de nuevo.

Nota: Una forma fácil de comprobar que el texto se convierte al juego de caracteres codificado 37 es utilizar IBM MQ Explorer. Examine la cola y muestre las propiedades del mensaje antes de que se recupere.

[4] Al recibir un mensaje JMS_IBM_CHARaque_SET es un nombre de página de códigos de Java Charset .

Contraste el fragmento de código de Figura 28 en la página 189 con el fragmento de código incorrecto de Figura 25 en la página 187. En el fragmento de código incorrecto, la serie de texto se convierte dos veces, una vez lo hace la aplicación y otra vez lo hace IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figura 25. Conversión de página de códigos incorrecta

El método `writeUTF` convierte texto de Unicode a 1208, UTF-8. La serie de texto tiene un prefijo de 2 bytes de longitud. La longitud máxima de la serie de texto es de 65534 bytes. El método `readUTF` lee un elemento de un mensaje escrito por el método `writeUTF`. Lee exactamente el número de bytes escritos por el método `writeUTF`.

Numérico

La codificación numérica predeterminada para un destino es `Native` (nativa). La constante de codificación `Native` para Java tiene el valor 273, `x'00000111'`, que es el mismo para todas las plataformas. En la recepción, los números del mensaje se transforman correctamente en primitivas Java numéricas. La transformación utiliza la codificación definida en el mensaje y el tipo devuelto por el método de lectura.

El método de envío convierte los números que se añaden a un mensaje mediante `set` y `write` a la codificación numérica definida para el destino. La codificación de destino se puede alterar temporalmente para un mensaje mediante una aplicación que establece la propiedad de mensaje `JMS_IBM_ENCODING`; por ejemplo:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

Los métodos numéricos `get` y `read` convierten los números del mensaje de la codificación numérica definida en el mensaje. Convierten los números al tipo especificado por el método `read` o `get`; consulte La propiedad ENCODING. Los métodos utilizan la codificación definida en `JMS_IBM_ENCODING`. La codificación se correlaciona desde `MQRFH2.Encoding`, a menos que el mensaje sea un mensaje de tipo MQ y no tenga `MQRFH2`. Si el mensaje es un mensaje de tipo MQ, sin `MQRFH2`, los métodos utilizan la codificación definida en `MQMD.Encoding`.

El ejemplo de Figura 30 en la página 189 muestra una aplicación que codifica un número en el formato de destino y lo envía en un `JMSStreamMessage`. Compare el ejemplo de Figura 30 en la página 189 con el ejemplo de Figura 31 en la página 190. La diferencia es que `JMS_IBM_ENCODING` se debe enviar en un `JMSBytesMessage`.

Nota: Una manera fácil de comprobar que el número se codifica correctamente es utilizar IBM MQ Explorer. Examine la cola y muestre las propiedades del mensaje antes de que se consuma.

Otro

Los métodos `boolean` codifican `true` y `false` como `x'01'` y `x'00'` en un `JMSByteMessage`, un `JMSStreamMessage` y un `JMSMapMessage`.

Los métodos UTF codifican y decodifican Unicode en series de texto UTF-8. Las series están limitadas a menos de 65536 caracteres y tienen como prefijo un campo de 2 bytes de longitud.

Los métodos de objeto encapsulan los tipos primitivos como objetos. Los tipos numéricos y de texto se codifican o convierten como si los tipos primitivos se hubieran leído o escrito utilizando los métodos numéricos y de texto.

Ninguna

Los métodos `readByte`, `readBytes`, `readUnsignedByte`, `writeByte` y `writeBytes` obtienen o transfieren bytes individuales, o matrices de bytes, entre la aplicación y el mensaje sin conversión. Los métodos `readChar` y `writeChar` obtienen y transfieren caracteres Unicode de 2 bytes entre la aplicación y el mensaje sin conversión.

Utilizando los métodos `readBytes` y `writeBytes`, la aplicación puede realizar su propia conversión de punto de código, como en "Envío y recepción de texto en un JMSBytesMessage" en la página 190.

IBM MQ no realizar ninguna conversión de la página de códigos en el cliente, debido a que el mensaje es un `JMSBytesMessage` y a que se utilizan los métodos `readBytes` y `writeBytes`. Sin embargo, si los bytes representan texto, asegúrese de que la página de códigos utilizada por la aplicación coincide con el juego de caracteres codificado del destino. Es posible que una salida de conversión de gestor de colas convierta de nuevo el mensaje. Otra posibilidad es que el programa cliente JMS de recepción pueda seguir el convenio de convertir cualquier matriz de bytes que representa texto del mensaje en series o caracteres utilizando la propiedad `JMS_IBM_CHARACTER_SET` del mensaje.

En este ejemplo, el cliente utiliza el juego de caracteres codificado de destino para su conversión:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Como alternativa, el cliente podría haber elegido una página de código y haber establecido luego el juego de caracteres codificado correspondiente en la propiedad `JMS_IBM_CHARACTER_SET` del mensaje. IBM MQ classes for Java utiliza `JMS_IBM_CHARACTER_SET` para establecer el campo `CodedCharacterSetId` en las propiedades JMS del `MQRFH2`, o en el descriptor de mensaje, `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
[5]
```

Si se escribe una matriz de bytes en un `JMSStringMessage` o un `JMSMapMessage`, IBM MQ classes for JMS no realiza la conversión de datos, ya que los bytes están escritos como datos hexadecimales, y no como texto, en el `JMSStringMessage` y el `JMSMapMessage`.

Si los bytes representan caracteres en su aplicación, debe tener en cuenta qué puntos de código leer y escribir en el mensaje. El código de Figura 26 en la página 188 sigue el convenio de utilizar el juego de caracteres codificado de destino. Si crea la serie utilizando el juego de caracteres predeterminado para la JVM, el contenido de bytes dependerá de la plataforma. Una JVM en Windows normalmente tiene un `Charset` predeterminado de `windows-1252`y AIX and Linux tiene `UTF-8`. Para el intercambio entre Windowsy AIX and Linux, debe seleccionar una página de códigos explícita para intercambiar texto como bytes.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Figura 26. Escritura de bytes que representan una serie en un JMS`StreamMessage` utilizando el juego de caracteres de destino

Ejemplos

[5] `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

Envío y recepción de un JMSTextmessage

Un mensaje de texto no puede contener texto en juegos de caracteres distintos. El ejemplo muestra texto en juegos de caracteres distintos, enviados en dos mensajes diferentes.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figura 27. Enviar texto de mensaje en el juego de caracteres definido por el destino

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figura 28. Enviar mensaje de texto en ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figura 29. Recibir mensaje de texto

Ejemplos de codificación

Ejemplos que muestran un número que se envía en la codificación definida para un destino. Tenga en cuenta que debe establecer la propiedad JMS_IBM_ENCODING de un JMSBytesMessage en el valor especificado para el destino.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

Figura 30. Envío de un número utilizando la codificación de destino en un JMSStreamMessage

```
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
```

Figura 31. Envío de un número utilizando la codificación de destino en un JMSBytesMessage

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 32 en la página 190 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir una serie de texto en un mensaje de bytes que contiene una combinación de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 33 en la página 190. Incluso para una serie con un número fijo de caracteres, la longitud de la representación de bytes podría ser mayor.

```
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);
```

Figura 32. Envío de una String en un JMSBytesMessage

```
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);
```

Figura 33. Recepción de una String de un JMSBytesMessage

Conceptos relacionados

Enfoques de conversión de mensajes JMS
Hay una serie de enfoques de conversión de datos disponibles para los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si su aplicación solo intercambia texto o intercambia mensajes únicamente con otras aplicaciones JMS, normalmente no se plantearía la conversión de datos. La conversión de datos se realiza automáticamente, mediante IBM MQ.

Conversión de datos de gestor de colas
La conversión de datos de gestor de colas siempre ha estado disponible para las aplicaciones que no son JMS y que reciben mensajes de clientes JMS. Los clientes de JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas, que es opcional.

Tareas relacionadas

Intercambio de un registro formateado con una aplicación no JMS
Siga los pasos sugeridos en esta tarea para diseñar y crear una salida de conversión de datos, y una aplicación cliente JMS que pueda intercambiar mensajes con una aplicación no JMS utilizando

JMSBytesMessage. El intercambio de un mensaje con formato con una aplicación no JMS se puede producir con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensaje JMS

La elección del tipo de mensaje afecta a la manera en que se enfoca la conversión de mensajes. La interacción de la conversión de mensajes y el tipo de mensaje se describe para los tipos de mensaje JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessage y JMSBytesMessage.

Conversión de datos de gestor de colas

La conversión de datos de gestor de colas siempre ha estado disponible para las aplicaciones que no son JMS y que reciben mensajes de clientes JMS. Los clientes de JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas, que es opcional.

El gestor de colas puede convertir datos numéricos y de caracteres en datos de mensaje utilizando los valores de CodedCharacterSetId, Encoding y Format establecidos para los datos de mensaje. Para aplicaciones que no son JMS, la capacidad de conversión siempre ha estado disponible estableciendo la opción GetMessageOption, GMO_CONVERT.

El gestor de colas puede convertir los mensajes que se envían a los clientes JMS . La conversión del gestor de colas se controla estableciendo la propiedad de destino, WMQ_RECEIVE_CONVERSION, en WMQ_RECEIVE_CONVERSION_QMGRo WMQ_RECEIVE_CONVERSION_CLIENT_MSG. La aplicación puede cambiar el valor de destino:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

O,

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 34. Habilitar la conversión de datos de gestor de colas

La conversión de datos de gestor de colas para un cliente JMS tiene lugar cuando el cliente llama a un método consumer.receive. Los datos de texto se transforman en UTF-8 (1208) de forma predeterminada. Los métodos de lectura y obtención subsiguientes decodifican el texto de los datos recibidos de UTF-8, creando primitivas de texto Java en su codificación Unicode interna. UTF-8 no es el único juego de caracteres de destino de la conversión de datos de gestor de colas. Puede optar por un CCSID distinto estableciendo la propiedad de destino WMQ_RECEIVE_CCSID.

Una aplicación también puede cambiar el valor de destino, por ejemplo estableciéndolo en 437, DOS-US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

O,

```
((MQDestination)destination).setReceiveCCSID(437);
```

Figura 35. Establecer el juego de caracteres codificado de destino para la conversión de gestor de colas

El motivo para cambiar WMQ_RECEIVE_CCSID está especializado; el CCSID elegido no supone ninguna diferencia para los objetos creados en la JVM. No obstante, es posible que algunas JVM, en determinadas plataformas, no puedan gestionar la conversión del CCSID del texto del mensaje en Unicode. La opción

le ofrece una elección de CCSID para cualquier texto proporcionado al cliente en el mensaje. Algunas plataformas de cliente JMS han tenido problemas con la entrega del texto de mensaje en UTF-8.

El código JMS es equivalente al texto en negrita en el código C en Figura 36 en la página 192,

```
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                     */
```

Figura 36. Fragmento de código de `amqsget0.c`

Nota:

La conversión del gestor de colas sólo se realiza en los datos de mensaje que tienen un formato IBM MQ conocido. `MQSTR` o `MQCIH` son ejemplos de formatos conocidos que están predefinidos. Un formato conocido puede ser también un formato definido por el usuario, siempre que haya proporcionado una salida de conversión de datos.

Los mensajes construidos como `JMSTextMessage`, `JMSMapMessage` y `JMSStreamMessage` tienen un formato `MQSTR` y pueden ser convertidos por el gestor de colas.

Conceptos relacionados

Enfoques de conversión de mensajes JMS
Hay una serie de enfoques de conversión de datos disponibles para los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si su aplicación solo intercambia texto o intercambia mensajes únicamente con otras aplicaciones JMS, normalmente no se plantearía la conversión de datos. La conversión de datos se realiza automáticamente, mediante IBM MQ.

Conversión y codificación de mensajes de cliente JMS
Se muestran los métodos que puede utilizar para realizar la conversión y codificación de mensajes de cliente JMS, con ejemplos de código de cada tipo de conversión.

“Invocación de la salida de conversión de datos” en la página 1010
Una salida de conversión de datos es una salida escrita por el usuario que recibe el control durante el procesamiento de una llamada MQGET.

Tareas relacionadas

Intercambio de un registro formateado con una aplicación no JMS
Siga los pasos sugeridos en esta tarea para diseñar y crear una salida de conversión de datos, y una aplicación cliente JMS que pueda intercambiar mensajes con una aplicación no JMS utilizando `JMSBytesMessage`. El intercambio de un mensaje con formato con una aplicación no JMS se puede producir con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensaje JMS

La elección del tipo de mensaje afecta a la manera en que se enfoca la conversión de mensajes. La interacción de la conversión de mensajes y el tipo de mensaje se describe para los tipos de mensaje JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` y `JMSBytesMessage`.

Intercambio de un registro formateado con una aplicación no JMS
Siga los pasos sugeridos en esta tarea para diseñar y crear una salida de conversión de datos, y una aplicación cliente JMS que pueda intercambiar mensajes con una aplicación no JMS utilizando `JMSBytesMessage`. El intercambio de un mensaje con formato con una aplicación no JMS se puede producir con o sin llamar a una salida de conversión de datos.

Antes de empezar

Es posible que pueda diseñar una solución más sencilla para intercambiar mensajes con una aplicación no JMS utilizando `JMSTextMessage`. Elimine esa posibilidad antes de seguir los pasos de esta tarea.

Acerca de esta tarea

Un cliente JMS es más fácil de escribir si no está implicado en los detalles de formatear los mensajes JMS intercambiados con otros clientes JMS. Siempre que el tipo de mensaje sea `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` o `JMSObjectMessage`, IBM MQ se hará cargo de los detalles del formateo del mensaje. IBM MQ se ocupa de las diferencias de las páginas de códigos y de la codificación numérica en las distintas plataformas.

Puede utilizar estos tipos de mensajes para intercambiar mensajes con aplicaciones no JMS. Para ello, debe comprender cómo IBM MQ classes for JMS construye estos mensajes. Es posible que pueda modificar la aplicación no JMS para interpretar los mensajes; consulte “Correlación de mensajes JMS en mensajes IBM MQ” en la página 154.

Una ventaja de usar estos tipos de mensaje consiste en que la programación de clientes JMS no depende del tipo de aplicación con la que se intercambian mensajes. Una desventaja es que podría requerir una modificación en otro programa, cosa que no siempre es posible.

Un enfoque alternativo es escribir una aplicación cliente JMS que se pueda ocupar de los formatos de mensaje existentes. A menudo, los mensajes existentes tienen un formato fijo y contienen una mezcla de datos, texto y números sin formato. Use los pasos de esta tarea y el ejemplo de cliente JMS en “Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage” en la página 196 como punto de partida para crear un cliente JMS que pueda intercambiar registros formateados con aplicaciones no JMS.

Procedimiento

1. Defina el diseño de registro, o utilice una de las clases de cabecera IBM MQ predefinidas.

   Para manejar cabeceras IBM MQ predefinidas, consulte Manejo de cabeceras de mensaje IBM MQ.

   Figura 37 en la página 194 es un ejemplo de un diseño de registro de longitud fija, definido por el usuario, que puede procesarse mediante la utilidad de conversión de datos.
2. Cree la salida de conversión de datos.

   Siga las instrucciones de Escritura de un programa de salida de conversión de datos para escribir una salida de conversión de datos.

   Para probar el ejemplo en “Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage” en la página 196, indique la salida de conversión de datos `MYRECORD`.
3. Escriba clases Java para encapsular el diseño de registro, y el diseño de envío y recepción. Dos posibles enfoques serían:

   - Escriba una clase en la que se lea y se escriba el `JMSBytesMessage` que contiene el registro; consulte “Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage” en la página 196.

- Grabe una clase que extienda `com.ibm.mq.header.Header` para definir la estructura de datos del registro; consulte Creación de clases para nuevos tipos de cabecera.

4. Decida el juego de caracteres codificado en el que se intercambiarán los mensajes.

   Consulte Elección de un enfoque para la conversión de mensajes: el receptor se hace cargo.

5. Configure el destino para intercambiar mensajes de tipo MQ, sin una cabecera de JMS MQRFH2.

   Tanto el destino de envío como el de recepción han de configurarse para intercambiar mensajes de tipo MQ. Puede utilizar el mismo destino para enviar y recibir.

   La aplicación puede sustituir la propiedad de cuerpo del mensaje del destino:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   En el ejemplo "Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage" en la página 196 se sustituye la propiedad de cuerpo de mensaje del destino, garantizando así que se envía un mensaje de estilo MQ.

6. Pruebe la solución con aplicaciones JMS y no JMS.

   Herramientas útiles para probar una salida de conversión de datos:

   - El programa de ejemplo `amqsgetc0.c` es útil para probar la recepción de un mensaje enviado por un cliente JMS. Consulte las modificaciones sugeridas para utilizar la cabecera de ejemplo, `RECORD.h`, en Figura 38 en la página 195. Con las modificaciones, `amqsgetc0.c` recibe un mensaje enviado por el ejemplo de cliente JMS, `TryMyRecord.java`; consulte "Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage" en la página 196.
   - El programa de examen de ejemplo IBM MQ, `amqsbcg0.c`, es útil para inspeccionar el contenido de la cabecera del mensaje, la cabecera JMS, MQRFH2 y el contenido del mensaje.
   - El programa **`rfhutil`** , anteriormente disponible en SupportPac IH03, permite que los mensajes de prueba se capturen y almacenen en archivos y, a continuación, se utilicen para dirigir flujos de mensajes. Los mensajes de salida también se pueden leer y visualizar en una serie de formatos. Los formatos incluyen dos tipos de XML, así como coinciden con un libro de copias COBOL. Los datos pueden estar en EBCDIC o ASCII. Se puede añadir una cabecera RFH2 al mensaje antes de que se envíe.

   Si intenta recibir mensajes utilizando el programa de ejemplo `amqsgetc0.c` modificado y le da un error con código de razón 2080, ompruebe si el mensaje tiene una MQRFH2. Las modificaciones presuponen que el mensaje se ha enviado a un destino que especifica que no hay MQRFH2.

Ejemplos

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                               MQLONG StructLength;
                               MQLONG Encoding;
                               MQLONG CodeCharSetId;
                               MQCHAR Format[8];
                               MQLONG Flags;
                               MQCHAR RecordData[32];
        };
```

Figura 37. RECORD.h

- Declare la estructura de datos 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);
```

- Modifique la llamada MQGET para que use RECORD,

  1. Antes de la modificación:

```
    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. Después de la modificación:

```
    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           */
```

- Cambie la sentencia de impresión,

  1. De:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. A:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

Figura 38. Modificación de amqsget0.c

Conceptos relacionados

Enfoques de conversión de mensajes JMS

Hay una serie de enfoques de conversión de datos disponibles para los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si su aplicación solo intercambia texto o intercambia mensajes únicamente con otras aplicaciones JMS, normalmente no se plantearía la conversión de datos. La conversión de datos se realiza automáticamente, mediante IBM MQ.

Conversión y codificación de mensajes de cliente JMS
Se muestran los métodos que puede utilizar para realizar la conversión y codificación de mensajes de cliente JMS, con ejemplos de código de cada tipo de conversión.

Conversión de datos de gestor de colas
La conversión de datos de gestor de colas siempre ha estado disponible para las aplicaciones que no son JMS y que reciben mensajes de clientes JMS. Los clientes de JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas, que es opcional.

Utilidad para crear código de salida de conversión

Referencia relacionada

Conversión y tipos de mensaje JMS
La elección del tipo de mensaje afecta a la manera en que se enfoca la conversión de mensajes. La interacción de la conversión de mensajes y el tipo de mensaje se describe para los tipos de mensaje JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage` y `JMSBytesMessage`.

Escritura de clases para encapsular un diseño de registro en un archivo `JMSBytesMessage`
La finalidad de esta tarea consiste en explorar, por ejemplo, cómo combinar la conversión de datos y un diseño de registro fijo en un `JMSBytesMessage`. En la tarea, cree algunas clases de Java para intercambiar una estructura de registro de ejemplo en un `JMSBytesMessage`. Puede modificar el ejemplo para escribir clases para intercambiar otras estructuras de registro.

Un `JMSBytesMessage` es la mejor opción del tipo de mensaje JMS para intercambiar registros de tipo de datos mixtos con programas que no son de programas JMS. No tiene ningún dato adicional insertado en el cuerpo del mensaje por el proveedor de JMS. Por lo tanto, es la mejor opción de tipo de mensaje utilizar si un programa cliente de JMS interactúa con un programa de IBM MQ existente. El principal reto al utilizar un `JMSBytesMessage` viene con en la correspondencia de la codificación con el juego de caracteres esperado por el otro programa. Una solución es crear una clase que encapsule el registro. Una clase que encapsula la lectura y la escritura de un `JMSBytesMessage`, para un tipo de registro específico, facilita el envío y la recepción de registros de formato fijo en un programa JMS. Al capturar los aspectos genéricos de la interfaz en una clase abstracta, gran parte de la solución se puede volver a utilizar para distintos formatos de registro. Se pueden implementar diferentes formatos de registro en clases que amplían la clase genérica abstracta.

Un enfoque alternativo consiste en ampliar la clase `com.ibm.mq.headers.Header`. La clase `Header` tiene métodos como, por ejemplo, `addMQLONG`, para crear un formato de registro de forma más declarativa. Una desventaja de utilizar la clase `Header` es que obtener y establecer los atributos utiliza una interfaz interpretativa más complicada. Ambos enfoques tienen como resultado la misma cantidad de código de aplicación.

Un `JMSBytesMessage` sólo puede encapsular un formato, además de una `MQRFH2`, en un mensaje, a menos que cada registro utilice el mismo formato, juego de caracteres codificado y codificación. El formato, la codificación y el juego de caracteres de un `JMSBytesMessage` son propiedades de todo el mensaje que sigue a la `MQRFH2`. El ejemplo se escribe suponiendo que un `JMSBytesMessage` contiene sólo un registro de usuario.

Antes de empezar

1. Su nivel de habilidad: debe estar familiarizado con la programación de Java y JMS. No se proporcionan instrucciones sobre cómo configurar el entorno de desarrollo de Java. Es muy útil haber escrito un programa para intercambiar un `JMSTextMessage`, `JMSStreamMessage` o `JMSMapMessage`. Así, podrá ver las diferencias al intercambiar un mensaje utilizando un `JMSBytesMessage`.
2. El ejemplo requiere IBM WebSphere MQ 7.0.
3. El ejemplo se ha creado utilizando la perspectiva Java del entorno de trabajo de Eclipse. Se necesita JRE 6.0 o superior. Puede utilizar la perspectiva de Java en IBM MQ Explorer para desarrollar y ejecutar las clases de Java. De forma alternativa, utilice su propio entorno de desarrollo de Java.
4. El uso de IBM MQ Explorer hace que crear el entorno de prueba y la depuración sea más sencillo que usar los programas de utilidad de línea de mandatos.

Acerca de esta tarea

Se le guía en la creación de dos clases: RECORD y MyRecord. De forma conjunta, estas dos clases encapsulan un registro de formato fijo. Tienen métodos para obtener y establecer atributos. El método get lee el registro de un JMSBytesMessage y el método put graba un registro en un JMSBytesMessage.

La finalidad de la tarea no es crear una clase de calidad de producción que se pueda volver a utilizar. Es posible que quiera utilizar los ejemplos en la tarea para empezar a trabajar en sus propias clases. La finalidad de la tarea es proporcionarle notas de orientación, principalmente sobre el uso de conjuntos de caracteres, formatos y codificación, cuando se utiliza un JMSBytesMessage. Se explican cada paso en la creación de las clases y se describen aspectos de la utilización de JMSBytesMessage, que a veces se pasan por alto.

La clase RECORD es abstracta y define algunos campos comunes para un registro de usuario. Los campos comunes se modelan en el diseño de cabecera de IBM MQ estándar de forma que tengan captación de atención, una versión y un campo de longitud. Se omiten los campos de codificación, juego de caracteres y formato, que se encuentran en muchas cabeceras de IBM MQ. Tras un formato definido por el usuario no puede haber otra cabecera. La clase MyRecord, que amplía la clase RECORD, lo hace extendiendo literalmente el registro con campos de usuario adicionales. Se puede procesar un JMSBytesMessage, creado por las clases, mediante la salida de conversión de datos del gestor de colas.

"Clases utilizadas para ejecutar el ejemplo" en la página 203 incluye una lista completa de RECORD y MyRecord. También incluye listados de las clases "scaffolding" adicionales para probar RECORD y MyRecord. Las clases adicionales son:

TryMyRecord
El programa principal para probar RECORD y MyRecord.

EndPoint
Una clase abstracta que encapsula la conexión de JMS, el destino y la sesión en una única clase. Su interfaz sólo responde a las necesidades de probar las clases RECORD y MyRecord. No es un patrón de diseño establecido para escribir aplicaciones de JMS.

Nota: La clase Endpoint incluye esta línea de código después de crear un destino:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

En la v7.0, desde la v7.0.1.5, es necesario activar la conversión del gestor de colas. De forma predeterminada está inhabilitado. En la v7.0, hasta la v7.0.1.4, la conversión del gestor de colas está habilitada de forma predeterminada, y esta línea de código causa un error.

MyProducer y MyConsumer
Clases que amplían EndPoint y crean un MessageConsumer y un MessageProducer, conectadas y listas para aceptar solicitudes.

Juntas, todas las clases forman una aplicación completa con la que puede crear y experimentar, para comprender cómo utilizar la conversión de datos en un JMSBytesMessage.

Procedimiento

1. Cree una clase abstracta para encapsular los campos estándar en una cabecera de IBM MQ, con un constructor predeterminado. Más adelante, amplíe la clase para adaptar la cabecera a sus requisitos.

```
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();
    }
```

Nota:

a. Los atributos, `structID` a `nextFormat`, se listan en el orden en que se establecen en una cabecera de mensaje de IBM MQ estándar.

b. Los atributos `format`, `messageEncoding` y `messageCharset`, describen la cabecera propiamente dicha, y no forman parte de la misma.

c. Debe decidir si desea almacenar el identificador de juego de caracteres codificados o el juego de caracteres del registro. Java utiliza juegos de caracteres y los mensajes de IBM MQ utilizan identificadores de juego de caracteres codificados. El código de ejemplo utiliza juegos de caracteres.

d. `int` se serializa en `MQLONG` mediante IBM MQ. `MQLONG` son 4 bytes.

2. Cree los métodos getter y setters para los atributos privados.

a) Cree o genere los métodos 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) Cree o genere los métodos 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. Cree un constructor para crear una instancia de RECORD a partir de un `JMSBytesMessage`.

```
  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());
  }
```

Nota:

a. `messageCharset` y `messageEncoding`, se capturan a partir de las propiedades del mensaje, ya que alteran temporalmente los valores establecidos para el destino. `format` no se

actualiza. El ejemplo no realiza la comprobación de errores. Si se llama al constructor `Record(BytesMessage)`, se supone que `JMSBytesMessage` es un mensaje de tipo RECORD. La línea "`setStructID(new String(structID, getMessageCharset()))`" establece la captación de atención.

b. Las líneas de código que completan los campos de deserialización de método en el mensaje, en orden, actualizan los conjuntos de valores predeterminados en la instancia RECORD.

4. Cree un método put para escribir los campos de cabecera en un `JMSBytesMessage`.

```
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;
}
```

Nota:

a. `MyProducer` encapsula JMS `Connection`, `Destination`, `Session` y `MessageProducer` en una sola clase. MyConsumer, que se utiliza más tarde, encapsula JMS `Connection`, `Destination`, `Session` y `MessageConsumer` en una sola clase.

b. Para un `JMSBytesMessage`, si la codificación no es `Native`, se debe establecer en el mensaje. La codificación de destino se copia en el atributo de codificación de mensajes, `JMS_IBM_CHARACTER_SET`, y se guarda como un atributo de la clase RECORD.

   i) "`setMessageEncoding(myProducer.getEncoding());`" invoca "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" para obtener la codificación de destino.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" establece la codificación del mensaje.

c. El juego de caracteres utilizado para transformar el texto en bytes se obtiene del destino y se guarda como un atributo de la clase RECORD. No se establece en el mensaje, porque no lo utiliza el IBM MQ classes for JMS al escribir un `JMSBytesMessage`.

   Llamadas "`messageCharset = myProducer.getCharset();`"

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

   Se obtiene el juego de caracteres de Java de un identificador de juego de caracteres codificado.

   " `CCSID.getCodepage(ccsid)` " está en el paquete `com.ibm.mq.headers`. El `ccsid` se obtiene de otro método en `MyProducer`, que consulta el destino.

```
public int getCCSID() throws JMSException {
    return (((MQDestination) destination)
            .getIntProperty(WMQConstants.WMQ_CCSID));
}
```

d. "myProducer.setMQClient(true);" altera temporalmente el valor de destino para el tipo de cliente, forzándolo a un IBM MQ MQI client. Es posible que prefiera omitir esta línea de código, ya que oculta un error de configuración administrativa.

"myProducer.setMQClient(true);" invoca:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

El código tiene el efecto secundario de establecer el estilo de cuerpo de IBM MQ en «no especificado», si se debe alterar temporalmente un valor de JMS.

Nota:

El IBM MQ classes for JMS graba el formato, codificación e identificador de juego de caracteres del mensaje en el descriptor de mensaje, MQMD, o en la cabecera de JMS, MQRFH2. Depende de si el mensaje tiene un cuerpo de estilo de IBM MQ. No establezca los campos MQMD manualmente.

Existe un método para establecer manualmente las propiedades del descriptor de mensaje. Utiliza las propiedades JMS_IBM_MQMD_*. Debe establecer la propiedad de destino, WMQ_MQMD_WRITE_ENABLED para establecer las propiedades de JMS_IBM_MQMD_*:

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Debe establecer la propiedad de destino, WMQ_MQMD_READ_ENABLED, para leer las propiedades.

Utilice JMS_IBM_MQMD_* solo si tiene control completo sobre la carga útil completa de mensajes. A diferencia de las propiedades de JMS_IBM_*, las propiedades de JMS_IBM_MQMD_* no controlan la forma en la que IBM MQ classes for JMS construye un mensaje JMS. Se pueden crear propiedades de descriptor de mensaje que entren en conflicto con las propiedades del mensaje JMS.

e. Las líneas de código que completan el método serializan los atributos en la clase como campos en el mensaje.

Los atributos de serie se rellenan con espacios en blanco. Las series se convierten en bytes utilizando el juego de caracteres definido para el registro, y se cortan en la longitud de los campos de mensaje.

5. Complete la clase añadiendo las importaciones.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Cree una clase para ampliar la clase RECORD para que incluya campos adicionales. Incluya un constructor predeterminado.

```
 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);
    }
```

Nota:

a. La subclase RECORD, MyRecord, personaliza la captador de atención, el formato y la longitud de cabecera.

7. Cree o genere los métodos getter y setter.

   a) Cree los métodos getter:

```
    public int getFlags() { return flags; }
   public String getRecordData() { return recordData; } .
```

   b) Cree los métodos setter:

```
    public void setFlags(int flags) {
       this.flags = flags; }
   public void setRecordData(String recordData) {
       this.recordData = recordData; }
}
```

8. Cree un constructor para crear una instancia de MyRecord a partir de un JMSBytesMessage.

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

Nota:

a. Los campos que componen la plantilla de mensaje estándar se leen en primer lugar mediante la clase RECORD.

b. El texto recordData se convierte en String utilizando la propiedad de juego de caracteres del mensaje.

9. Cree un método estático para obtener un mensaje de un consumidor y crear una nueva instancia de MyRecord.

```
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
           MQDataException, IOException {
       BytesMessage message = (BytesMessage) myConsumer.receive();
       return new MyRecord(message);
    }
```

Nota:

a. En el ejemplo, por brevedad, se llama al constructor MyRecord(BytesMessage) desde el método get estático. Por lo general, puede separar la recepción del mensaje de la creación de una nueva instancia de MyRecord.

10. Cree un método put para añadir los campos de cliente a un JMSBytesMessage que contenga una cabecera de mensaje.

```
    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;
    }
```

Nota:

a. Las llamadas de método en el código serializan los atributos de la clase MyRecord como campos en el mensaje.

- El atributo `recordData` `String` se rellena con espacios en blanco, se convierte en bytes utilizando el juego de caracteres definido para el registro y se recorta a la longitud de los campos `RecordData`.

11. Complete la clase añadiendo las sentencias include.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Resultados

- El resultado de ejecutar la clase `TryMyRecord`:
  - Envío de mensaje en el juego de caracteres codificado 37 y utilizando una salida de conversión de gestor de colas:

```
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
```

  - Envío de mensaje en el juego de caracteres codificado 37 y sin usar una salida de conversión de gestor de colas:

```
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
```

- El resultado de la modificación de la clase `TryMyRecord` para que no reciba el mensaje y, en su lugar, lo reciba utilizando el ejemplo `amqsget0.c` modificado. El ejemplo modificado acepta un registro con formato; consulte Figura 38 en la página 195 en “Intercambio de un registro formateado con una aplicación no JMS” en la página 193.
  - Envío de mensaje en el juego de caracteres codificado 37 y utilizando una salida de conversión de gestor de colas:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - Envío de mensaje en el juego de caracteres codificado 37 y sin usar una salida de conversión de gestor de colas:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
```

```
no more messages
Sample AMQSGET0 end
```

Para probar el ejemplo y hacer pruebas con páginas de códigos diferentes y con una salida de conversión de datos. Cree las clases Java, configure IBM MQ y ejecute el programa principal, `TryMyRecord`; consulte “#unique_196/unique_196_Connect_42_Try” en la página 203.

1. Configure IBM MQ y JMS para ejecutar el ejemplo. Las instrucciones son para ejecutar el ejemplo en Windows.
   a. Crear un gestor de colas

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. Crear una cola

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Crear un directorio JNDI

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. Cambiar al directorio bin de JMS

   El programa de administración de JMS se debe ejecutar desde aquí. La vía de acceso es *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Cree las definiciones siguientes de JMS en un archivo denominado `JMSQM1Q1.txt`

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. Ejecute el programa JMSAdmin para crear los recursos JMS

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. Puede crear, modificar y examinar las definiciones que ha creado utilizando IBM MQ Explorer.
3. Ejecute `TryMyRecord`.

Clases utilizadas para ejecutar el ejemplo

Las clases listadas en los siguientes bloques de código también están disponibles en un archivo comprimido. Descargue jm25529_.zip o jm25529_.tar.gz.

`TryMyRecord`

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
```

```
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

RECORD

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
```

```
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
```

```
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
```

```
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
```

```
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
```

```
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
```

```
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

V 9.3.0 JM 3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Creación y configuración de fábricas de conexiones y destinos

Una aplicación IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging puede crear fábricas de conexiones y destinos recuperándolos como objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface), utilizando las extensiones de IBM JMS o utilizando las extensiones de IBM MQ JMS . Una aplicación también puede utilizar las extensiones de IBM JMS o las extensiones de IBM MQ JMS para establecer las propiedades de fábricas y destinos de conexiones.

Las fábricas de conexiones y los destinos son puntos de partida en el flujo de la lógica de una aplicación JMS o Jakarta Messaging . Una aplicación utiliza un objeto ConnectionFactory para crear una conexión a un servidor de mensajería y utiliza un objeto Queue o Topic como destino para enviarle mensajes o como origen desde el que recibir mensajes. Por tanto, una aplicación debe crear al menos una fábrica de conexiones y uno o varios destinos. Al haber creado una fábrica de conexiones o un destino, la aplicación podría tener que configurar el objeto estableciendo una o varias de sus propiedades.

En resumen, una aplicación puede crear y configurar fábricas de conexiones y destinos mediante los procedimientos siguientes:

Utilización de JNDI para recuperar objetos administrados
Un administrador puede utilizar la herramienta de administración de IBM MQ JMS tal como se describe en Configuración de objetos JMS y Jakarta Messaging utilizando las herramientas de administración, o IBM MQ Explorer tal como se describe en Configuración de objetos JMS 2.0 utilizando IBM MQ Explorer, para crear y configurar fábricas de conexiones y destinos como objetos administrados en un espacio de nombres JNDI. Una aplicación puede entonces recuperar los objetos administrados del espacio de nombres JNDI. Después de haber recuperado un objeto administrado, la aplicación puede, si es necesario, establecer o cambiar una o más de sus propiedades utilizando las extensiones de IBM JMS o las extensiones de IBM MQ JMS.

Nota: V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, no puede administrar JNDI utilizando IBM MQ Explorer. La administración JNDI está soportada por la variante Jakarta Messaging 3.0 de **JMSAdmin**, que es **JMS30Admin**.

utilización de las extensiones de IBM JMS
Una aplicación puede utilizar las extensiones de IBM JMS para crear destinos y fábricas de conexiones dinámicamente durante el tiempo de ejecución. En primer lugar, la aplicación crea un objeto JmsFactoryFactory y luego utiliza métodos de este objeto para crear fábricas de conexiones y destinos. Al haber creado una fábrica de conexiones o un destino, la aplicación puede utilizar métodos heredados de la interfaz JmsPropertyContext para establecer sus propiedades. La aplicación también puede utilizar un identificador universal de recursos (URI) para especificar una o varias propiedades de un destino cuando se crea un destino.

utilización de las extensiones de IBM MQ JMS
Una aplicación también puede utilizar las extensiones de IBM MQ JMS para crear destinos y fábricas de conexiones dinámicamente durante el tiempo de ejecución. La aplicación utiliza los constructores proporcionados para crear fábricas de conexiones y destinos. Después de crear una fábrica de conexiones o un destino, la aplicación puede utilizar métodos del objeto para establecer sus propiedades. La aplicación también puede utilizar un URI para especificar una o varias propiedades de un destino cuando se crea un destino.

Tareas relacionadas

Configuración de recursos JMS y Jakarta Messaging

Utilización de JNDI para recuperar objetos administrados en una aplicación JMS o Jakarta Messaging
Para recuperar objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación JMS o Jakarta Messaging debe crear un contexto inicial y, a continuación, utilizar el método lookup () para recuperar los objetos.

Para que una aplicación pueda recuperar objetos administrados de un espacio de nombres JNDI, primero un administrador debe crear los objetos administrados.

JMS 2.0 Para JMS 2.0, el administrador puede utilizar la herramienta de administración de IBM MQ JMS , **JMSAmin**, o IBM MQ Explorer para crear y mantener objetos administrados en un espacio de nombres JNDI. Para obtener más información, consulte Configuración de fábricas de conexiones y destinos en un espacio de nombres JNDI.

V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, no puede administrar JNDI utilizando IBM MQ Explorer. La administración JNDI está soportada por la variante Jakarta Messaging 3.0 de **JMSAdmin**, que es **JMS30Admin**.

Normalmente, un servidor de aplicaciones proporciona su propio repositorio para objetos administrados y sus propias herramientas para crear y mantener los objetos.

Para recuperar objetos administrados de un espacio de nombres JNDI, primero una aplicación debe crear un contexto inicial, tal como se muestra en el siguiente ejemplo:

V 9.3.0 JM 3.0 V 9.3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

En este código las variables de String, `url` y `icf` tienen los significados siguientes:

url
El localizador uniforme de recursos (URL) del servicio de directorio. El URL puede tener uno de los formatos siguientes:

- `ldap://`*`hostname`*`/`*`contextName`* , para un servicio de directorio basado en un servidor LDAP
- `file:/`*`directoryPath`* , para un servicio de directorio basado en el sistema de archivos local

icf
El nombre de clase de la fábrica de contexto inicial, que puede ser uno de los valores siguientes:

- `com.sun.jndi.ldap.LdapCtxFactory`, para un servicio de directorio basado en un servidor LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`, para un servicio de directorio basado en el sistema de archivos local

Observe que algunas combinaciones de un paquete JNDI y un proveedor de servicios LDAP (Lightweight Directory Access Protocol) pueden hacer que se produzca el error 84 de LDAP. Para resolver este problema, inserte la siguiente línea de código antes de realizar la llamada a InitialDirContext():

```
environment.put(Context.REFERRAL, "throw");
```

Después de obtener un contexto inicial, la aplicación puede recuperar los objetos administrados de un espacio de nombres JNDI utilizando el método lookup(), tal como se muestra en el siguiente ejemplo:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
```

```
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Este código recuperar los objetos siguientes de un espacio de nombres basado en LDAP:

- Un objeto ConnectionFactory enlazado al nombre myCF
- Un objeto Queue enlazado al nombre myQ
- Un objeto Topic enlazado al nombre myT

Para obtener más información sobre el uso de JNDI, consulte la documentación de JNDI proporcionada por Oracle Corporation.

Tareas relacionadas

Configuración de objetos JMS 2.0 utilizando IBM MQ Explorer

Configuración de objetos JMS y Jakarta Messaging utilizando las herramientas de administración

Configuración de recursos JMS 2.0 en WebSphere Application Server

utilización de las extensiones de IBM JMS

IBM MQ classes for JMS (JMS 2.0) y IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) contienen cada uno un conjunto de extensiones funcionalmente idénticas a la API JMS denominada extensiones IBM JMS . Una aplicación puede utilizar estas extensiones para crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución y para establecer las propiedades de objetos IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging . Las extensiones se pueden utilizar con cualquier proveedor de mensajería.

Las extensiones de IBM JMS son un conjunto de interfaces y clases en los paquetes siguientes:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Para Jakarta Messaging 3.0, estos paquetes están en `com.ibm.jakarta.client.jar`.

JMS 2.0 Para JMS 2.0, estos paquetes están en `com.ibm.mqjms.jar` o `com.ibm.mq.allclient.jar`.

Estas extensiones proporcionan las funciones siguientes:

- Un mecanismo para crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, en lugar de recuperarlos como objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface)
- Un conjunto de métodos para establecer las propiedades de objetos IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging
- Un conjunto de clases de excepciones con métodos para obtener información detallada sobre un problema
- Un conjunto de métodos para controlar el rastreo
- Un conjunto de métodos para obtener información de versión sobre IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging

Para crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, y establecer y obtener sus propiedades, las extensiones de IBM JMS proporcionan un conjunto alternativo de interfaces a las extensiones de IBM MQ JMS . Sin embargo, mientras que las extensiones de IBM MQ JMS son específicasdel proveedor de mensajería IBM MQ, las extensiones de IBM JMS no son específicas de IBM MQ y se pueden utilizar con cualquier proveedor de mensajería dentro de la arquitectura en capas descrita en Arquitectura de clases de IBM MQ para JMS.

La interfaz com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) o com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) contiene las definiciones de constantes que una aplicación puede utilizar al establecer las propiedades de objetos IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utilizando las extensiones de IBM JMS . La interfaz contiene

constantes para el proveedor de mensajería de IBM MQ y constantes de JMS que son independientes de cualquier proveedor de mensajería.

Los ejemplos de código siguientes presuponen que las siguientes sentencias de importación se incluyen en la clase Java :

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0

```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Creación de fábricas de conexiones y destinos

Antes de que una aplicación pueda crear destinos y fábricas de conexiones utilizando las extensiones de IBM JMS, en primer lugar, debe crear un objeto JmsFactoryFactory. Para crear un objeto JmsFactoryFactory, la aplicación llama al método getInstance() de la clase JmsFactoryFactory, tal como se muestra en el ejemplo siguiente:

V 9.3.0 JM 3.0 V 9.3.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

El parámetro de la llamada getInstance() es una constante que identifica el proveedor de mensajería de IBM MQ como proveedor de mensajería elegido. A continuación, la aplicación puede utilizar el objeto JmsFactoryFactory para crear fábricas de conexiones y destinos.

Para crear un fábrica de conexiones, la aplicación llama al método createConnectionFactory() del objeto JmsFactoryFactory, tal como se muestra en el ejemplo siguiente:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Esta sentencia crea un objeto JmsConnectionFactory con los valores predeterminados para todas sus propiedades, lo que significa que la aplicación se conecta al gestor de colas predeterminado en modalidad de enlaces. Si desea que una aplicación se conecte en modalidad de cliente, o bien que se conecte a un gestor de colas que no sea el gestor de colas predeterminado, la aplicación debe establecer las propiedades adecuadas del objeto JmsConnectionFactory antes de crear la conexión. Para obtener información sobre cómo conseguirlo consulte el apartado “Establecimiento de las propiedades de objetos de IBM MQ classes for JMS” en la página 215.

La clase JmsFactoryFactory también contiene métodos para crear fábricas de conexiones de los tipos siguientes:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Para crear un objeto Queue, la aplicación llama al método createQueue() de la clase JmsFactoryFactory, tal como se muestra en el ejemplo siguiente:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Esta sentencia crea un objeto JmsQueue con los valores predeterminados para todas sus propiedades. El objeto representa una cola de IBM MQ denominada Q1 que pertenece al gestor de colas local. Esta cola puede ser una cola local, una cola de alias o una definición de cola remota.

El método createQueue() también puede aceptar un identificador universal de recursos (URI) de cola como parámetro. Un URI de cola es una serie que especifica el nombre de una cola de IBM MQ y, opcionalmente, el nombre del gestor de colas que es propietario de la cola, y una o más propiedades del objeto JmsQueue. La sentencia siguiente contiene un ejemplo de un URI de cola:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

El objeto JmsQueue creado por esta sentencia representa una cola IBM MQ denominada Q2 que es propiedad del gestor de colas QM2, y todos los mensajes enviados a este destino son persistentes y tienen una prioridad de 5. Para obtener más información sobre los URI de cola, consulte “Identificadores uniformes de recursos (URI)” en la página 228. Para conocer un procedimiento alternativo para establecer las propiedades de un objeto JmsQueue, consulte “Establecimiento de las propiedades de objetos de IBM MQ classes for JMS” en la página 215.

Para crear un objeto Topic, una aplicación puede utilizar el método createTopic() del objeto JmsFactoryFactory, tal como se muestra en el ejemplo siguiente:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Esta sentencia crea un objeto JmsTopic con los valores predeterminados para todas sus propiedades. El objeto representa un tema llamado Sport/Football/Results.

El método createTopic() también puede aceptar un URI de tema como parámetro. Un URI de tema es una serie que especifica el nombre de un tema y, opcionalmente, una o varias propiedades del objeto JmsTopic. Las sentencias siguientes contienen un ejemplo de un URI de tema:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

El objeto JmsTopic creado por estas sentencias representa un tema denominado Sport/Tennis/Results, y todos los mensajes enviados a este destino no son persistentes y tienen una prioridad de 0. Para obtener más información sobre los URI de tema, consulte “Identificadores uniformes de recursos (URI)” en la página 228. Para conocer un procedimiento alternativo para establecer las propiedades de un objeto JmsTopic, consulte “Establecimiento de las propiedades de objetos de IBM MQ classes for JMS” en la página 215.

Después de que una aplicación haya creado una fábrica de conexiones o un destino, ese objeto sólo puede utilizarse con el proveedor de mensajería seleccionado.

Establecimiento de las propiedades de objetos de IBM MQ classes for JMS

Para establecer las propiedades de objetos IBM MQ classes for JMS utilizando las extensiones de IBM JMS, una aplicación utiliza los métodos de la interfaz com.ibm.msg.client.JmsPropertyContext. De forma similar, para establecer las propiedades de los objetos IBM MQ classes for Jakarta Messaging utilizando las extensiones de IBM JMS , una aplicación utiliza los métodos de la interfaz com.ibm.msg.jakarta.client.JmsPropertyContext .

Para cada tipo de datos de Java, la interfaz JmsPropertyContext contiene un método para establecer el valor de una propiedad con ese tipo de datos y un método para obtener el valor de una propiedad con ese tipo de datos. Por ejemplo, una aplicación invoca el método setIntProperty() para establecer una propiedad con un valor entero e invoca el método getIntProperty() para obtener una propiedad con un valor entero.

Las instancias de clases de los paquetes com.ibm.mq.jms y com.ibm.mq.jakarta.jms heredan los métodos de las interfaces de contexto JmsPropertycorrespondientes. Por tanto, una aplicación puede utilizar estos métodos para establecer las propiedades de los objetos MQConnectionFactory, MQQueue y MQTopic.

Cuando una aplicación crea un objeto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , las propiedades con valores predeterminados se establecen automáticamente. Cuando una aplicación establece una propiedad, el nuevo valor sustituye cualquier valor anterior que tuviera la propiedad. Después de haber establecido una propiedad, esta no se puede suprimir, pero su valor se puede cambiar.

Si una aplicación intenta establecer una propiedad en un valor que no es un valor válido para la propiedad, IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging emite una excepción JMSException. Si una aplicación intenta obtener una propiedad que no se ha establecido, el comportamiento es como el descrito en la especificación JMS. IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging emiten una excepción de excepción NumberFormatpara tipos de datos primitivos y devuelven nulo para tipos de datos referenciados.

Además de las propiedades predefinidas de un objeto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , una aplicación puede establecer sus propias propiedades. IBM MQ classes for JMS y IBM MQ classes for Jakarta Messagingignoran estas propiedades definidas por la aplicación.

Para obtener más información sobre las propiedades de los objetos IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging , consulte Propiedades de objetos IBM MQ classes for JMS.

El código siguiente es un ejemplo de cómo establecer propiedades utilizando las extensiones de IBM JMS. El código establece cinco propiedades de una fábrica de conexiones.

```
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");
```

El efecto de establecer estas propiedades es que la aplicación se conecta al gestor de colas QM1 en la modalidad de cliente, utilizando un canal MQI denominado QM1.SVR. El gestor de colas se ejecuta en un sistema cuyo nombre de host es HOST1 y el proceso de escucha del gestor de colas está a la escucha en el número de puerto 1415. Esta conexión y otras conexiones del gestor de colas asociadas tienen el nombre de aplicación "Mi aplicación" asociado a ellas.

Nota: Los gestores de colas que se ejecutan en las plataformas z/OS no permiten establecer nombres de aplicación, por lo que este valor no se tiene en cuenta.

La interfaz JmsPropertyContext también contiene el método setObjectProperty(), que una aplicación puede utilizar para establecer propiedades. El segundo parámetro del método es un objeto que encapsula el valor de la propiedad. Por ejemplo, el código siguiente crea un objeto Integer que encapsula el entero 1415 y, a continuación, invoca setObjectProperty() para establecer la propiedad PORT de una fábrica de conexiones en el valor 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Por tanto, este código es equivalente a la sentencia siguiente:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

En cambio, el método getObjectProperty() devuelve un objeto que encapsula el valor de una propiedad.

Conversión implícita de un valor de propiedad de un tipo de datos a otro

Cuando una aplicación utiliza un método de la interfaz de contexto JmsPropertypara establecer u obtener la propiedad de un objeto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , el valor de la propiedad se puede convertir implícitamente de un tipo de datos a otro.

Por ejemplo, la sentencia siguiente establece la propiedad PRIORITY del objeto JmsQueue q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

La propiedad PRIORITY tiene un valor entero y, por tanto, la llamada setStringProperty() convierte implícitamente la serie "5" (el valor de origen) en el entero 5 (el valor de destino), que entonces pasa a ser el valor de la propiedad PRIORITY.

En cambio, la sentencia siguiente obtiene la propiedad PRIORITY del objeto JmsQueue q1:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

La llamada getStringProperty() convierte implícitamente el entero 5 (el valor de origen), que es el valor de la propiedad PRIORITY, en la serie "5" (el valor de destino).

Las conversiones soportadas por IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging se muestran en la Tabla 34 en la página 217.

Tabla 34. Conversiones soportadas de un tipo de datos a otro

Tipo de datos de origen	**Tipos de datos de destino soportados**
boolean	Serie
byte	int, long, short, String
char	Serie
double	Serie
float	double, String
int	long, String
Entero largo	Serie
short	int, long, String
Serie	boolean, byte, double, float, int, long, short

Las normas generales que rigen las conversiones soportadas son las siguientes:

- Los valores numéricos pueden convertirse de un tipo de datos a otro siempre que no se pierdan datos durante la conversión. Por ejemplo, un valor con el tipo de datos `int` puede convertirse en un valor con el tipo de datos `long`, pero no puede convertirse en un valor con el tipo de datos `short`.
- Un valor de cualquier tipo de datos puede convertirse en una serie.
- Una serie puede convertirse en un valor de cualquier otro tipo de datos (excepto `char`) siempre que la serie esté en el formato correcto para la conversión. Si una aplicación intenta convertir una serie que no está en el formato correcto, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging emiten una excepción de excepción NumberFormat.
- Si una aplicación intenta una conversión que no está soportada, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging emiten una excepción de excepción MessageFormat.

Las reglas específicas para convertir un valor de un tipo de datos a otro son las siguientes:

- Al convertir un valor booleano en una serie, el valor `true` se convierte en la serie "true", mientras que el valor `false` se convierte en la serie "false".

- Al convertir una serie en un valor booleano, la serie "true" (no sensible a mayúsculas y minúsculas) se convierte en `true`, mientras que la serie "false" (no sensible a mayúsculas y minúsculas) se convierte en `false`. Cualquier otra serie se convierte en `false`.
- Al convertir una serie en un valor con el tipo de datos `byte`, `int`, `long` o `short`, la serie debe tener el formato siguiente:

  [*espacios en blanco*][*signo*]*dígitos*

  El significado de los componentes de la serie es el siguiente:

  espacios en blanco
  Caracteres en blanco iniciales opcionales.

  signo
  Un signo más (+) o un signo menos (-) opcional.

  dígitos
  Una secuencia contigua de dígitos (0-9). Al menos debe proporcionarse un dígito.

  Después de la secuencia de dígitos, la serie puede contener otros caracteres que no sean dígitos, pero la conversión se detiene cuando se llega al primero de estos caracteres. Se da por sentado que la serie representa un entero decimal.

  Si la serie no tiene el formato correcto, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging emiten una excepción de excepción NumberFormat.
- Al convertir una serie en un valor con el tipo de datos `double` o `float`, la serie debe tener el formato siguiente:

  [*espacios en blanco*][*signo*]*dígitos*[*carácter_e*[*signo_e*]*dígitos_e*]

  El significado de los componentes de la serie es el siguiente:

  espacios en blanco
  Caracteres en blanco iniciales opcionales.

  signo
  Un signo más (+) o un signo menos (-) opcional.

  dígitos
  Una secuencia contigua de dígitos (0-9). Al menos debe proporcionarse un dígito.

  e_car
  Un carácter exponente, que puede ser *E* o *e*.

  e_signo
  Un signo más (+) o un signo menos (-) opcional para el exponente.

  e_dígitos
  Una secuencia contigua de dígitos (0-9) para el exponente. Al menos debe proporcionarse un dígito si la serie contiene un carácter de exponente.

  Después de la secuencia de dígitos o de los caracteres opcionales que representan un exponente, la serie puede contener otros caracteres que no sean dígitos, pero la conversión se detiene cuando se llega al primero de estos caracteres. Se da por supuesto que la serie representa un número de coma flotante decimal con un exponente que es una potencia de 10.

  Si la serie no tiene el formato correcto, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging emiten una excepción de excepción NumberFormat.
- Al convertir un valor numérico (incluido un valor con el tipo de datos `byte`) en una serie, el valor se convierte en la representación de la serie del valor como número decimal, no la serie que contiene el carácter ASCII para ese valor. Por ejemplo, el entero 65 se convierte a la serie "65", no la serie "A".

Establecimiento de más de una propiedad en una sola llamada

La interfaz JmsPropertyContext también contiene el método setBatchProperties(), que una aplicación puede utilizar para establecer más de una propiedad en una sola llamada. El parámetro del método es un objeto Map que encapsula un conjunto de pares nombre-valor de propiedades.

Por ejemplo, el código siguiente utiliza el método setBatchProperties() para establecer las mismas cinco propiedades de una fábrica de conexiones, tal como se muestra en el apartado "Establecimiento de las propiedades de objetos de IBM MQ classes for JMS" en la página 215. El código crea una instancia de la clase HashMap, que implementa la interfaz Map.

```
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);
```

Tenga en cuenta que el segundo parámetro del método Map.put() debe ser un objeto. Por tanto, un valor de propiedad con un tipo de dato de primitivos debe estar encapsulado en un objeto o representado por una serie, tal como se muestra en el ejemplo.

El método setBatchProperties() valida las propiedades. Si el método setBatchProperties() no puede establecer una propiedad porque, por ejemplo, su valor no es válido, no se establece ninguna de las propiedades especificadas.

Nombres y valores de propiedades

Si una aplicación utiliza los métodos de la interfaz de contexto JmsPropertyadecuada para establecer y obtener las propiedades de los objetos IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , la aplicación puede especificar los nombres y valores de las propiedades de cualquiera de las formas siguientes. En los ejemplos siguientes se muestra cómo establecer la propiedad PRIORITY del objeto JmsQueue q1 para que un mensaje enviado a la cola tenga la prioridad especificada en la llamada send().

Utilizando nombres y valores de propiedades que están definidos como constantes en la interfaz com.ibm.msg.client.wmq.WMQConstants

La sentencia siguiente es un ejemplo de cómo especificar los nombres y valores de propiedades de esa manera:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Utilizando nombres y valores de propiedades que pueden utilizarse en identificadores uniformes de recursos (URI) de cola y de tema

La sentencia siguiente es un ejemplo de cómo especificar los nombres y valores de propiedades de esa manera:

```
q1.setIntProperty("priority", -2);
```

Sólo los nombres y valores de propiedades de destinos se pueden especificar de esta manera.

Utilizando los nombre de propiedad y los valores reconocidos por la herramienta de administración IBM MQ JMS

La sentencia siguiente es un ejemplo de cómo especificar los nombres y valores de propiedades de esa manera:

```
q1.setStringProperty("PRIORITY", "APP");
```

La forma corta del nombre de propiedad también es aceptable, tal como se muestra en la sentencia siguiente:

```
q1.setStringProperty("PRI", "APP");
```

Cuando una aplicación obtiene una propiedad, el valor devuelto depende de la manera en que la aplicación especifica el nombre de la propiedad. Por ejemplo, si una aplicación especifica la constante WMQConstants.WMQ_PRIORITY como nombre de la propiedad, el valor devuelto es el entero -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

Se devuelve el mismo valor si la aplicación especifica la serie "priority" como nombre de propiedad:

```
int n2 = getIntProperty("priority");
```

Sin embargo, si la aplicación especifica la serie "PRIORITY" o "PRI" como nombre de propiedad, el valor devuelto es la serie "APP":

```
String s1 = getStringProperty("PRI");
```

Internamente, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging almacenan nombres y valores de propiedad como los valores literales definidos en la interfaz WMQConstants coincidente. Este es el formato canónico definido para los nombres y valores de propiedad. Como regla general, si una aplicación establece propiedades utilizando una de las otras dos formas de especificar nombres y valores de propiedad, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging tienen que convertir los nombres y valores del formato de entrada especificado al formato canónico. De forma similar, si una aplicación obtiene propiedades utilizando una de las otras dos formas de especificar nombres y valores de propiedad, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging deben convertir los nombres del formato de entrada especificado al formato canónico y convertir los valores del formato canónico al formato de salida necesario. Tener que efectuar estas conversiones puede tener implicaciones en el rendimiento.

Los nombres de propiedad y los valores devueltos por las excepciones, en los archivos de rastreo, o en el registro de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , siempre están en el formato canónico.

Utilización de la interfaz Map

La interfaz JmsPropertyContext amplía la interfaz java.util.Map. Por lo tanto, una aplicación puede utilizar los métodos de la interfaz Map para acceder a las propiedades de un objeto IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging .

Por ejemplo, el código siguiente visualiza los nombres y valores de todas las propiedades de una fábrica de conexiones. El código sólo utiliza los métodos de la interfaz Map para obtener los nombres y valores de las propiedades.

```
// 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));
}
```

La utilización de métodos de la interfaz Map no elude las validaciones o conversiones de propiedades.

utilización de las extensiones de IBM MQ JMS
IBM MQ classes for JMS contiene un conjunto de extensiones en la API JMS denominadas las extensiones de IBM MQ JMS. Una aplicación puede utilizar estas extensiones para crear dinámicamente destinos y

fábricas de conexiones en tiempo de ejecución, y para establecer las propiedades de las fábricas de conexiones y los destinos.

IBM MQ classes for JMS contiene un conjunto de clases en los paquetes com.ibm.jms y com.ibm.mq.jms. Estas clases implementan las interfaces de JMS y contienen las extensiones de IBM MQ JMS. En los ejemplos de código que siguen a continuación se presupone que estos paquetes se han importado mediante las sentencias siguientes:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Una aplicación puede utilizar las extensiones de IBM MQ JMS para realizar las funciones siguientes:

- Crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, en lugar de recuperarlos como objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface)
- Establecer las propiedades de fábricas de conexiones y destinos

Creación de fábricas de conexiones

Para crear una fábrica de conexiones, una aplicación puede utilizar el constructor MQConnectionFactory, tal como se muestra en el siguiente ejemplo:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Esta sentencia crea un objeto MQConnectionFactory con los valores predeterminados para todas las propiedades, lo que significa que la aplicación se conecta al gestor de colas predeterminado en modalidad de enlaces. Si desea que una aplicación se conecte en modalidad de cliente o se conecte a un gestor de colas distinto del gestor de colas predeterminado, la aplicación debe establecer las propiedades adecuadas del objeto MQConnectionFactory antes de crear la conexión. Para obtener información sobre cómo conseguirlo consulte el apartado “Establecimiento de las fábricas de conexiones” en la página 221.

Una aplicación puede crear fábricas de conexiones de los tipos siguientes de un modo similar:

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

Establecimiento de las fábricas de conexiones

Una aplicación puede establecer las propiedades de una fábrica de conexiones invocando los métodos apropiados de la fábrica de conexiones. La fábrica de conexiones puede ser un objeto administrado o un objeto creado dinámicamente en tiempo de ejecución.

En el caso del código siguiente:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Este código crea un objeto MQConnectionFactory y a continuación, establece cinco propiedades del objeto. El efecto de establecer estas propiedades es que la aplicación se conecta al gestor de colas QM1

en modalidad de cliente mediante un canal MQI denominado QM1.SVR. El gestor de colas se ejecuta en un sistema cuyo nombre de host es HOST1 y el proceso de escucha del gestor de colas está a la escucha en el número de puerto 1415.

Una aplicación que utiliza una conexión en tiempo real con un intermediario sólo puede utilizar el estilo de mensajería de publicación/suscripción. No puede utilizar el estilo de mensajería punto a punto.

Sólo son válidas determinadas combinaciones de propiedades de una fábrica de conexiones. Para obtener información sobre qué combinaciones son válidas, consulte Dependencias entre propiedades de objetos IBM MQ classes for JMS.

Para obtener más información sobre las propiedades de una fábrica de conexiones y los métodos utilizados para establecer sus propiedades, consulte Propiedades de objetos IBM MQ classes for JMS.

Creación de destinos

Para crear un objeto Queue, una aplicación puede utilizar el constructor MQQueue, tal como se muestra en el ejemplo siguiente:

```
MQQueue q1 = new MQQueue("Q1");
```

Esta sentencia crea un objeto MQQueue con los valores predeterminados para todas las propiedades. El objeto representa una cola de IBM MQ denominada Q1 que pertenece al gestor de colas local. Esta cola puede ser una cola local, una cola de alias o una definición de cola remota.

Una forma alternativa del constructor MQQueue tiene dos parámetros, tal como se muestra en el ejemplo siguiente:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

El objeto MQQueue creado por esta sentencia representa una cola de IBM MQ denominada Q2 que es propiedad del gestor de colas QM2. El gestor de colas identificado de esta manera puede ser el gestor de colas local o un gestor de colas remoto. Si es un gestor de colas remoto, IBM MQ debe estar configurado para que, cuando la aplicación envíe un mensaje a este destino, WebSphere MQ pueda encaminar el mensaje desde el gestor de colas local al gestor de colas remoto.

El constructor MQQueue también puede aceptar un identificador uniforme de recursos (URI) como parámetro. Un URI de cola es una serie que especifica el nombre de una cola de IBM MQ y, opcionalmente, el nombre del gestor de colas que es propietario de la cola, y una o más propiedades del objeto MQQueue. La sentencia siguiente contiene un ejemplo de un URI de cola:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

El objeto MQQueue creado por esta sentencia representa una cola IBM MQ denominada Q3 que es propiedad del gestor de colas QM3, y todos los mensajes enviados a este destino son persistentes y tienen una prioridad de 5. Para obtener más información sobre los URI de cola, consulte “Identificadores uniformes de recursos (URI)” en la página 228. Para obtener un método alternativo para establecer las propiedades de un objeto MQQueue, consulte el apartado “Establecimiento de las propiedades de los destinos” en la página 223.

Para crear un objeto Topic, una aplicación puede utilizar el constructor MQTopic, tal como se muestra en el siguiente ejemplo:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Esta sentencia crea un objeto MQTopic con el valor predeterminado para todas las propiedades. El objeto representa un tema llamado Sport/Football/Results.

El constructor MQTopic también puede aceptar un URI de tema como parámetro. Un URI de tema es una serie que especifica el nombre de un tema, y opcionalmente, una o más propiedades del objeto MQTopic. La siguiente sentencia contiene un ejemplo de un URI de tema:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

El objeto MQTopic creado por esta sentencia representa un tema denominado Sport/Tennis/Results, y todos los mensajes enviados a este destino no son persistentes y tienen una prioridad de 0. Para obtener más información sobre los URI de tema, consulte "Identificadores uniformes de recursos (URI)" en la página 228. Para obtener un método alternativo para establecer las propiedades de un objeto MQTopic, consulte el apartado "Establecimiento de las propiedades de los destinos" en la página 223.

Establecimiento de las propiedades de los destinos

Una aplicación puede establecer las propiedades de un destino invocando los métodos apropiados del destino. El destino puede ser un objeto administrado o un objeto creado dinámicamente en tiempo de ejecución.

En el caso del código siguiente:

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Este código crea un objeto MQQueue y a continuación, establece dos propiedades del objeto. Como resultado del establecimiento de estas propiedades, todos los mensajes enviados al destino son permanentes y tienen una prioridad de 5.

Una aplicación puede establecer las propiedades del objeto MQTopic de un modo similar, tal como se muestra en el siguiente ejemplo:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Este código crea un objeto MQTopic y a continuación, establece dos propiedades del objeto. Como resultado del establecimiento de estas propiedades, todos los mensajes enviados al destino son no permanentes y tienen una prioridad de 0.

Para obtener más información sobre las propiedades de un destino y los métodos utilizados para establecer sus propiedades, consulte Propiedades de objetos IBM MQ classes for JMS.

Linux AIX *Conexión a IBM MQ desde una aplicación JMS*

Para crear una conexión, una aplicación JMS utiliza un objeto **ConnectionFactory** para crear un objeto **Connection** y, a continuación, inicia la conexión.

Para JMS 2.0 y posteriores, las aplicaciones normalmente se conectan a un proveedor de mensajería utilizando un objeto **ConnectionFactory** y el método `createContext()` .

En versiones anteriores de JMS, primero tenía que utilizar `createConnection` para crear un objeto **Connection** y, a continuación, iniciar la llamada de conexión `getSession()` para crear un objeto **Session** que pudiera llevar a cabo operaciones de mensajería.

Un objeto **JMSContext** encapsula de forma efectiva los objetos **Connection** y **Session** . Si desea utilizar el enfoque tradicional y crear la conexión y los objetos de sesión directamente, consulte "Creación de una sesión en una aplicación de JMS" en la página 224 y "Creación de una sesión en una aplicación de JMS" en la página 225.

Para crear un objeto **JMSContext** , una aplicación utiliza el método `createContext()` de un objeto **ConnectionFactory** , tal como se muestra en el ejemplo siguiente:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

Cuando se crea una conexión JMS , el IBM MQ classes for JMS crea un descriptor de conexión (Hconn) e inicia una conversación con el gestor de colas.

Nota: Observe que el ID de proceso de aplicación se utiliza como identificador de usuario predeterminado que debe pasarse al gestor de colas. Si la aplicación se ejecuta en la modalidad de transporte de cliente, este ID de proceso debe existir, con las autorizaciones pertinentes, en el servidor. Si desea que se utilice una identidad diferente, utilice el método `createConnection(username,password)` .

V 9.3.5 Este mecanismo también se puede utilizar para proporcionar una señal de autenticación, consulte Obtención de una señal de autenticación del emisor de señal elegido.

JMS 1.0 *Creación de una sesión en una aplicación de JMS*

Para crear una conexión en JMS 1.0, una aplicación JMS utiliza un objeto ConnectionFactory para crear un objeto Connection y, a continuación, inicia la conexión.

Para crear un objeto Connection, una aplicación utiliza el método createConnection() de un objeto ConnectionFactory, tal como se muestra en el siguiente ejemplo:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Cuando se crea una conexión de JMS, IBM MQ classes for JMS crea un descriptor de contexto de conexión (Hconn) e inicia una conversación con el gestor de colas.

La interfaz QueueConnectionFactory y la interfaz TopicConnectionFactory heredan cada una el método createConnection() de la interfaz ConnectionFactory. Por consiguiente, utilice el método createConnection() para crear un objeto específico de dominio, tal como se muestra en el ejemplo siguiente:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Este fragmento de código crea un objeto QueueConnection. Ahora una aplicación puede realizar una operación independiente de dominio en este objeto o una operación que sólo es aplicable al dominio punto a punto. Sin embargo, si la aplicación intenta realizar una operación que sólo es aplicable al dominio de publicación/suscripción, se emite la excepción con el siguiente mensaje:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Esto se debe a que la conexión se creó desde una fábrica de conexiones específica de dominio.

Nota: Observe que el ID de proceso de aplicación se utiliza como identificador de usuario predeterminado que debe pasarse al gestor de colas. Si la aplicación se ejecuta en la modalidad de

transporte de cliente, este ID de proceso debe existir, con las autorizaciones pertinentes, en el servidor. Si desea que se utilice una identidad diferente, utilice el método createConnection(username, password).

La especificación JMS indica que una conexión se crea en el estado `stopped` . Hasta que se inicie una conexión, un consumidor de mensaje que esté asociado a la conexión no puede recibir ningún mensaje. Para iniciar una conexión, una aplicación utiliza el método start() de un objeto Connection, tal como se muestra en el siguiente ejemplo:

```
connection.start();
```

V 9.3.5 Este mecanismo también se puede utilizar para proporcionar una señal de autenticación, consulte Obtención de una señal de autenticación del emisor de señal elegido.

JMS 1.0 *Creación de una sesión en una aplicación de JMS*

Para crear una sesión en JMS 1.0, una aplicación JMS utiliza el método createSession() de un objeto Connection.

El método createSession() tiene dos parámetros:

1. Un parámetro que especifica si la sesión es transaccional o no.
2. Un parámetro que especifica la modalidad de acuse de recibo de la sesión

Por ejemplo, el código siguiente crea una sesión que no es transaccional y su modalidad de acuse de recibo es AUTO_ACKNOWLEDGE:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Cuando se crea una sesión de JMS, IBM MQ classes for JMS crea un descriptor de conexión (Hconn) e inicia una conversación con el gestor de colas.

Un objeto Session y cualquier objeto MessageProducer o MessageConsumer creado a partir de él no pueden ser utilizados conjuntamente por varias hebras de una aplicación multihebra. La forma más simple de asegurarse de que estos objetos no se utilicen simultáneamente es crear un objeto Session separado para cada hebra.

V 9.3.5 Este mecanismo también se puede utilizar para proporcionar una señal de autenticación, consulte Obtención de una señal de autenticación del emisor de señal elegido.

Sesiones transaccionales en aplicaciones JMS

Las aplicaciones JMS pueden ejecutar transacciones locales creando primero una sesión transaccional. Una aplicación puede confirmar o retrotraer una transacción.

Las aplicaciones JMS pueden ejecutar transacciones locales. Una transacción local es una transacción que implica cambios sólo en los recursos del gestor de colas al que está conectada la aplicación. Para ejecutar transacciones locales, una aplicación debe primero crear una sesión transaccional llamando al método createSession() de un objeto Connection, especificando como parámetro que la sesión es transaccional. Por consiguiente, todos los mensajes enviados y recibidos dentro de la sesión se agrupan en una secuencia de transacciones. Una transacción finaliza cuando la aplicación confirma o retrotrae los mensajes que ha enviado y recibido desde que empezó la transacción.

Para confirmar una transacción, una aplicación llama al método commit() del objeto Session. Cuando se confirma una transacción, todos los mensajes enviados en la transacción pasan a estar disponibles para su entrega a otras aplicaciones, y todos los mensajes recibidos en la transacción reciben el acuse de recibo, de forma que el servidor de mensajería no los intenta volver a entregar a la aplicación. En el dominio punto a punto, el servidor de mensajería también elimina los mensajes recibidos de sus colas.

Para retrotraer una transacción, una aplicación llama al método rollback() del objeto Session. Cuando una transacción se retrotrae, el servidor de mensajería descarta todos los mensajes enviados en la

transacción y todos los mensajes recibidos en la transacción pasan a estar disponibles para volverlos a entregar. En el dominio punto a punto, los mensajes que se han recibido se vuelven a colocar en sus colas y pasar a ser visibles de nuevo para otras aplicaciones.

Una nueva transacción se inicia automáticamente cuando una aplicación crea una sesión transaccional o llama al método commit() o rollback(). Por lo tanto, una sesión con transacción siempre tiene una transacción activa.

Cuando una aplicación cierra una sesión con transacción, se produce una retrotracción implícita. Cuando una aplicación cierra una conexión, se produce una retrotracción implícita para todas las sesiones con transacción de la conexión.

Si una aplicación finaliza sin cerrar una conexión, se produce también una retrotracción implícita para todas las sesiones transaccionales de la conexión.

Una transacción está incluida íntegramente en una sesión con transacción. Una transacción no puede abarcar sesiones. Esto significa que no es posible para una aplicación enviar y recibir mensajes en dos o más sesiones con transacción y, después, confirmar o retrotraer todas estas acciones como una sola transacción.

Modalidades de acuse de recibo de las sesiones de JMS
Cada sesión que no es una sesión con transacción tiene una modalidad de acuse de recibo que determina cómo se acusa recibo de los mensajes recibidos por la aplicación. Hay disponibles tres modalidades de acuse de recibo y la selección de la modalidad de acuse de recibo afecta al diseño de la aplicación.

Si una sesión no es una sesión con transacción, la forma en la que se acusa recibo de los mensajes recibidos por la aplicación se determina mediante la modalidad de acuse de recibo de la sesión. Las tres modalidades de acuse de recibo se describen en los párrafos siguientes:

AUTO_ACKNOWLEDGE

La sesión acusa recibo automáticamente de cada mensaje recibido por la aplicación.

Si los mensajes se entregan de forma síncrona a la aplicación, la sesión acusa recibo de cada mensaje cada vez que se completa una llamada Receive. Si los mensajes se entregan asíncronamente, la sesión acusa recibo de un mensaje cada vez que una llamada al método onMessage() de un escucha de mensajes se completa correctamente.

Si la aplicación recibe un mensaje correctamente, pero una anomalía impide el acuse de recibo, el mensaje pasa a estar disponible para volverse a entregar. Por lo tanto, la aplicación debe poder manejar un mensaje que se vuelve a entregar.

DUPS_OK_ACKNOWLEDGE

La sesión acusa recibo de los mensajes recibidos por la aplicación en momentos que selecciona.

El uso de esta modalidad de acuse de recibo reduce la cantidad de trabajo que debe realizar la sesión, pero una anomalía que impide el acuse de recibo de mensaje podría provocar que más de un mensaje pasara a estar disponible para una nueva entrega. Por lo tanto, la aplicación debe poder manejar mensajes que se vuelven a entregar.

Restricción: En las modalidades AUTO_ACKNOWLEDGE y DUPS_OK_ACKNOWLEDGE, JMS no permite que una aplicación emita una excepción no controlada en un escucha de mensajes. Esto significa que siempre se proporciona acuse de recibo de los mensajes cuando el escucha de mensajes devuelve el control, sin importar si se ha procesado correctamente, siempre que los errores no sean graves y no impidan que la aplicación pueda continuar. Si necesita un control más preciso del acuse de recibo de los mensajes, utilice las modalidades CLIENT_ACKNOWLEDGE o transaccional, que dan a la aplicación un control completo de las funciones de acuse de recibo.

CLIENT_ACKNOWLEDGE

La aplicación acusa recibo de los mensajes que recibe llamando al método Acusar recibo de la clase Message.

La aplicación acusa recibo de cada mensaje de forma individual, o puede recibir un lote de mensajes y llamar al método Acusar recibo solo para el último mensaje que recibe. Cuando se llama al método

Acusar recibo, se acusará recibo de todos los mensajes recibidos desde la última vez que se llamó al método.

Junto con cualquiera de estas modalidades de acuse de recibo, una aplicación puede detener y reiniciar la entrega de mensajes en una sesión llamando al método Recuperar de la clase Session. Los mensajes recibidos pero no reconocidos anteriormente se vuelven a entregar. Sin embargo, podría ser que no se entregaran en la misma secuencia en la que se habían entregada anteriormente. Mientras tanto, podrían haber llegado los mensajes con prioridad superior y algunos de los mensajes originales podrían haber caducado. En el dominio punto a punto, algunos de los mensajes originales podrían haber sido consumidos por otra aplicación.

Una aplicación puede determinar si un mensaje se está volviendo a entregar examinando el contenido del campo de cabecera JMSRedelivered del mensaje. Para ello, la aplicación llama al método getJMSRedelivered() de la clase Message.

Creación de destinos en una aplicación de JMS

En lugar de recuperar destinos como objetos administrados desde un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación de JMS puede utilizar una sesión para crear destinos de forma dinámica en tiempo de ejecución. ´ Una aplicación puede utilizar un identificador uniforme de recursos (URI) para identificar una cola o tema de IBM MQ y, opcionalmente, especificar una o más propiedades de un objeto Queue o Topic.

Utilización de una sesión para crear objetos Queue

Para crear un objeto Queue, una aplicación puede utilizar el método createQueue() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Este código crea un objeto Queue con los valores predeterminados para todas las propiedades. El objeto representa una cola de IBM MQ denominada Q1 que pertenece al gestor de colas local. Esta cola puede ser una cola local, una cola de alias o una definición de cola remota.

El método createQueue() también acepta un URI de cola como parámetro. Un URI de cola es una serie que especifica el nombre de una cola de IBM MQ y, opcionalmente, el nombre del gestor de colas que es propietario de la cola, y una o más propiedades del objeto Queue. La sentencia siguiente contiene un ejemplo de un URI de cola:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

El objeto Queue creado por esta sentencia representa una cola IBM MQ denominada Q2 que es propiedad de un gestor de colas denominado QM2, y todos los mensajes enviados a este destino son persistentes y tienen una prioridad de 5. El gestor de colas identificado de esta manera puede ser el gestor de colas local o un gestor de colas remoto. Si es un gestor de colas remoto, IBM MQ debe estar configurado para que, cuando la aplicación envíe un mensaje a este destino, WebSphere MQ pueda encaminar el mensaje desde el gestor de colas local al gestor de colas QM2. Para obtener más información sobre los URI, consulte el apartado “Identificadores uniformes de recursos (URI)” en la página 228.

Observe que el parámetro en el método createQueue() contiene información específica del proveedor. Por consiguiente, si se utiliza el método createQueue() para crear un objeto Queue en lugar de recuperar un objeto Queue como un objeto administrado desde un espacio de nombres JNDI, resultado podría ser que la aplicación fuera menos portable.

Una aplicación puede crear un objeto TemporaryQueue utilizando el método createTemporaryQueue() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Aunque se utiliza una sesión para crear una cola temporal, el ámbito de una cola temporal es la conexión que se ha utilizado para crear la sesión. Cualquiera de las sesiones de la conexión puede crear productores y consumidores de mensajes para la cola temporal. La cola temporal permanece hasta que la conexión finalice o la aplicación suprima explícitamente la cola temporal utilizando el método TemporaryQueue.delete(), lo que suceda antes.

Cuando una aplicación crea una cola temporal, IBM MQ classes for JMS crea una cola dinámica en el gestor de colas al que está conectada la aplicación. La propiedad TEMPMODEL de la fábrica de conexiones especifica el nombre de la cola de modelos que se utiliza para crear la cola dinámica y la propiedad TEMPQPREFIX de la fábrica de conexiones especifica el prefijo que se utiliza para formar el nombre de la cola dinámica.

Utilización de una sesión para crear objetos Topic

Para crear un objeto Topic, una aplicación puede utilizar el método createTopic() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Este código crea un objeto Topic con los valores predeterminados para todas las propiedades. El objeto representa un tema llamado Sport/Football/Results.

El método createTopic() también acepta un URI de tema como parámetro. Un URI de tema es una serie que especifica el nombre de un tema y, opcionalmente, una o más propiedades del objeto Topic. El código siguiente contiene un ejemplo de URI de tema:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

El objeto Topic creado por este código representa un tema llamado Sport/Tennis/Results, y todos los mensajes enviados a este destino no son persistentes y tienen una prioridad de 0. Para obtener más información sobre los URI de tema, consulte “Identificadores uniformes de recursos (URI)” en la página 228.

Observe que el parámetro del método createTopic() contiene información específica del proveedor. Por consiguiente, si se utiliza el método createTopic() para crear un objeto Topic en lugar de recuperar un objeto Topic como un objeto administrado desde un espacio de nombres JNDI, el resultado podría ser que la aplicación fuera menos portable.

Una aplicación puede crear un objeto TemporaryTopic utilizando el método createTemporaryTopic() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Aunque se utiliza una sesión para crear un tema temporal, el ámbito de un tema temporal es la conexión que se ha utilizado para crear la sesión. Cualquiera de las sesiones de la conexión puede crear productores y consumidores de mensajes para el tema temporal. El tema temporal permanece hasta que la conexión finalice o la aplicación suprima explícitamente el tema temporal utilizando el método TemporaryTopic.delete(), lo que suceda antes.

Cuando una aplicación crea un tema temporal, IBM MQ classes for JMS crea un tema con un nombre que empieza con los caracteres TEMP/*tempTopicPrefix*, donde *tempTopicPrefix* es el valor de la propiedad TEMPTOPICPREFIX de la fábrica de conexiones.

Identificadores uniformes de recursos (URI)

Un URI de cola es una serie que especifica el nombre de una cola de IBM MQ y, opcionalmente, el nombre del gestor de colas que es propietario de la cola, y una o más propiedades del objeto Queue creado por

la aplicación. Un URI de tema es una serie que especifica el nombre de un tema y, opcionalmente, una o más propiedades del objeto Topic creado por la aplicación.

Un URI de cola tiene el formato siguiente:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Un URI de tema tiene el formato siguiente:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Las variables con formatos tienen los significados siguientes:

nombre_gestor_colas
Nombre del gestor de colas que es propietario de la cola identificada por el URI.

El gestor de colas puede ser el gestor de colas local o un gestor de colas remoto. Si es un gestor de colas remoto, IBM MQ debe estar configurado para que, cuando la aplicación envíe un mensaje a la cola, WebSphere MQ pueda encaminar el mensaje desde el gestor de colas local al gestor de colas remoto.

Si no se especifica ningún nombre, se presupone que es el gestor de colas local.

qName
El nombre de la cola de IBM MQ.

La cola puede ser una cola local, una cola alias o una definición de cola remota.

Para las reglas para crear nombres de cola, consulte Reglas para designar objetos IBM MQ.

topicName
El nombre del tema.

Para las reglas para crear nombres de tema, consulte Reglas para designar objetos IBM MQ. Evite utilizar los caracteres comodín +, #, * y ? en nombres de temas. Los nombres de tema que contengan esos caracteres pueden producir resultados inesperados cuando un usuario se suscribe al tema. Consulte Combinación de series de tema.

***propertyName1*, *propertyName2*, ...**
Los nombres de las propiedades del objeto Queue o Topic creado por la aplicación. La Tabla 35 en la página 230 lista los nombres de propiedad válidos que se pueden utilizar en un URI.

Si no se especifica ninguna propiedad, el objeto Queue o Topic tiene los valores predeterminados para todas las propiedades.

***propertyValue1*, *propertyValue2*, ...**
Los valores de las propiedades del objeto Queue o Topic creado por la aplicación. La Tabla 35 en la página 230 lista los valores de propiedad válidos que se pueden utilizar en un URI.

Los corchetes ([]) denotan un componente opcional, y los puntos suspensivos (...) significan que la lista de pares nombre-valor de la propiedad, si está presente, puede contener uno o más pares nombre-valor.

La Tabla 35 en la página 230 lista los nombres de propiedad válidos que se pueden utilizar en los URI de cola y de tema. Aunque la herramienta de administración de IBM MQ JMS utiliza constantes simbólicas para los valores de propiedades, los URI no pueden contener constantes simbólicas.

Tabla 35. Nombres de propiedad y valores válidos para utilizar en los URI de cola y de tema

Nombre de propiedad	**Descripción**	**Valores válidos**
CCSID	Determina cómo se representan los datos de caracteres contenidos en el cuerpo de un mensaje cuando IBM MQ classes for JMS reenvía el mensaje al destino	• Cualquier identificador de juego de caracteres codificados que sea compatible con IBM MQ.
codificación	Determina cómo se representan los datos numéricos contenidos en el cuerpo de un mensaje cuando IBM MQ classes for JMS reenvía el mensaje al destino	• Cualquier valor válido del campo *Encoding* contenido en un descriptor de mensaje de IBM MQ.
caducidad	Tiempo de vida de los mensajes enviados al destino	• -2 - Tal como se ha especificado en la llamada send() o, en ausencia de esto, el tiempo de vida predeterminado del productor de mensajes. • 0 - Un mensaje enviado al destino no caduca nunca • Un entero positivo que especifica el tiempo de vida en milisegundos.
multicast	Valor de multidifusión para un tema cuando se utiliza una conexión en tiempo real con un intermediario	La lista siguiente contiene los valores válidos. Cada valor tiene asociado el valor correspondiente de la propiedad MULTICAST, tal como se utiliza en la herramienta de administración de IBM MQ JMS. Para obtener una descripción de la propiedad MULTICAST y sus valores válidos, consulte Propiedades de objetos IBM MQ classes for JMS. • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED

Tabla 35. Nombres de propiedad y valores válidos para utilizar en los URI de cola y de tema (continuación)

Nombre de propiedad	Descripción	Valores válidos
persistence	La persistencia de los mensajes enviados al destino	• -2 - Tal como se ha especificado en la llamada send() o si no se ha especificado en la llamada send(), la persistencia predeterminada del productor de mensajes. • -1 - Tal como se ha especificado en el atributo *DefPersistence* de la cola o tema de IBM MQ. • 1 - No persistente • 2 - Persistente • 3 - Equivalente al valor HIGH para la propiedad PERSISTENCE tal como se utiliza en la herramienta de administración de IBM MQ JMS. Para obtener una explicación de este valor, consulte "Mensajes persistentes de JMS" en la página 260.
priority	La prioridad de los mensajes enviados al destino	• -2 - Tal como se ha especificado en la llamada send() o si no se ha especificado en la llamada send(), la prioridad predeterminada del productor de mensajes. • -1 - Tal como se ha especificado en el atributo *DefPriority* de la cola o tema de IBM MQ. • Un entero comprendido dentro del rango 0-9 que especifica la prioridad de los mensajes enviados al destino.
targetClient	Si los mensajes enviados al destino contienen una cabecera MQRFH2	• 0 - Los mensajes contienen una cabecera MQRFH2. • 1 - Los mensajes no contienen una cabecera MQRFH2.

Por ejemplo, el URI siguiente identifica una cola de IBM MQ denominada Q1 que es propiedad del gestor de colas local. Un objeto Queue creado con este URI tiene todas sus propiedades establecidas en los valores predeterminados.

```
queue:///Q1
```

El URI siguiente identifica una cola de IBM MQ denominada Q2 que es propiedad de un gestor de colas denominado QM2. Todos los mensajes enviados a este destino tienen una prioridad de 6. Las propiedades restantes del objeto Queue creadas utilizando este URI tienen sus valores predeterminados.

```
queue://QM2/Q2?priority=6
```

El siguiente URI identifica un tema titulado Sport/Athletics/Results. Todos los mensajes enviados a este destino son no persistentes y tienen una prioridad de 0. Las propiedades restantes del objeto de tema creado utilizando este URI tienen sus valores predeterminados.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Envío de mensajes en una aplicación de JMS

Para que una aplicación de JMS pueda enviar mensajes a un destino, primero debe crear un objeto MessageProducer para el destino. Para enviar un mensaje al destino, la aplicación crea un objeto Message y luego llama al método send() del objeto MessageProducer.

Una aplicación utiliza un objeto MessageProducer para enviar mensajes. Normalmente, una aplicación crea un objeto MessageProducer para un destino específico, que puede ser una cola o un tema, de modo que todos los mensajes enviados mediante el productor de mensajes se envían al mismo destino. Por consiguiente, para que una aplicación pueda crear un objeto MessageProducer, primero debe crear un objeto Queue o Topic. Para obtener información sobre cómo crear un objeto Queue o Topic, consulte los temas siguientes:

- "Utilización de JNDI para recuperar objetos administrados en una aplicación JMS o Jakarta Messaging" en la página 211
- "utilización de las extensiones de IBM JMS" en la página 213
- "utilización de las extensiones de IBM MQ JMS" en la página 220
- "Creación de destinos en una aplicación de JMS" en la página 227

Para crear un objeto MessageProducer, una aplicación utiliza el método createProducer() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
MessageProducer producer = session.createProducer(destination);
```

El parámetro `destination` es un objeto Queue o un objeto Topic que la aplicación ha creado anteriormente.

Para que una aplicación pueda enviar un mensaje, debe crear un objeto Message. El cuerpo de un mensaje contiene los datos de aplicación, y JMS define cinco tipos de cuerpo de mensaje:

- Bytes
- Correlación
- Objeto
- Corriente
- Texto

Cada tipo de cuerpo de mensaje tiene su propia interfaz de JMS, que es una subinterfaz de la interfaz de mensajes, y un método en la interfaz de sesión para crear un mensaje con ese tipo de cuerpo. Por ejemplo, la interfaz de un mensaje de texto se denomina TextMessage y una aplicación utiliza el método createTextMessage() de un objeto Session para crear un mensaje de texto, tal como se muestra en la sentencia siguiente:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Para obtener más información sobre los mensajes y cuerpos de mensaje, consulte "Mensajes de JMS" en la página 149.

Para enviar un mensaje, una aplicación utiliza el método send() de un objeto MessageProducer, tal como se muestra en el siguiente ejemplo:

```
producer.send(outMessage);
```

Una aplicación puede utilizar el método send() para enviar mensajes en cualquiera de los dominios de mensajería. La naturaleza del destino determina qué dominio de mensajería se utiliza. No obstante, TopicPublisher, la subinterfaz de MessageProducer que es específica del dominio de publicación/suscripción también tiene un método publsih(), que se puede utilizar en lugar del método send(). Los dos métodos son funcionalmente similares.

Una aplicación puede crear un objeto MessageProducer sin destino especificado. En este caso, la aplicación debe especificar el destino cuando invoca el método send().

Si una aplicación envía un mensaje dentro de una transacción, el mensaje no se entrega a su destino hasta que se confirma la transacción. Esto significa que una aplicación no puede enviar un mensaje y recibir una respuesta al mensaje dentro de la misma transacción.

Un destino se puede configurar de forma que cuando una aplicación le envía mensajes, IBM MQ classes for JMS reenvía el mensaje y devuelve el control a la aplicación sin determinar si el gestor de colas has recibido el mensaje correctamente. Esto a veces se denomina *transferencia asíncrona*. Para obtener más información, consulte "Transferencia asíncrona de mensajes en IBM MQ classes for JMS" en la página 329.

Recepción de mensajes en una aplicación JMS

Una aplicación utiliza un consumidor de mensajes para recibir mensajes. Un suscriptor de tema duradero es un consumidor de mensajes que recibe todos los mensajes enviados a un destino, incluidos los enviados mientras el consumidor está inactivo. Una aplicación puede seleccionar qué mensajes desea recibir utilizando un selector de mensajes y puede recibir mensajes asíncronamente utilizando un escucha de mensajes.

Una aplicación utiliza un objeto MessageConsumer para recibir mensajes. Una aplicación crea un objeto MessageConsumer para un destino específico, que puede ser una cola o un tema para que todos los mensajes recibidos mediante el consumidor de mensajes se reciban desde el mismo destino. Por consiguiente, para que una aplicación pueda crear un objeto MessageConsumer, primero debe crear un objeto Queue o Topic. Para obtener información sobre cómo crear un objeto Queue o Topic, consulte los temas siguientes:

- "Utilización de JNDI para recuperar objetos administrados en una aplicación JMS o Jakarta Messaging" en la página 211
- "utilización de las extensiones de IBM JMS" en la página 213
- "utilización de las extensiones de IBM MQ JMS" en la página 220
- "Creación de destinos en una aplicación de JMS" en la página 227

Para crear un objeto MessageConsumer, una aplicación utiliza el método createConsumer() de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
MessageConsumer consumer = session.createConsumer(destination);
```

El parámetro `destination` es un objeto Queue o un objeto Topic que la aplicación ha creado anteriormente.

A continuación, la aplicación utiliza el método receive() del objeto MessageConsumer para recibir un mensaje del destino, tal como se muestra en milisegundos en el siguiente ejemplo:

```
Message inMessage = consumer.receive(1000);
```

El parámetro de la llamada receive() especifica con qué frecuencia en milisegundos el método espera hasta que llegue un mensaje adecuado si no hay ningún mensaje disponible de forma inmediata. Si se omite este parámetro, la llamada se bloquea de modo indefinido hasta que llegue un mensaje adecuado. Si no desea que la aplicación espere un mensaje, utilice en su lugar el método receiveNoWait().

El método receive() devuelve un mensaje de un tipo específico. Por ejemplo, cuando una aplicación recibe un mensaje de texto, el objeto devuelto por la llamada receive() es un objeto TextMessage.

Sin embargo, el tipo declarado de objeto devuelto por una llamada receive() es un objeto Message. Por consiguiente, para poder extraer los datos del cuerpo de un mensaje que se acaba de recibir, la aplicación debe difundirse desde la clase Message hasta la subclase más específica, como por ejemplo TextMessage. Si el tipo de mensaje no es conocido, la aplicación puede utilizar el operador `instanceof` para determinar el tipo. Es siempre recomendable que una aplicación determine el tipo de un mensaje antes de realizar la difusión de mensajes para que los errores se puedan tratar de forma ordenada.

El código siguiente utiliza el operador `instanceof` y muestra cómo extraer los datos del cuerpo de un mensaje de texto:

```
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");
}
```

Si una aplicación envía un mensaje dentro de una transacción, el mensaje no se entrega a su destino hasta que se confirma la transacción. Esto significa que una aplicación no puede enviar un mensaje y recibir una respuesta al mensaje dentro de la misma transacción.

Si un consumidor de mensajes recibe mensajes desde un destino que está configurado para la lectura anticipada, los mensajes no persistentes que se encuentran en el almacenamiento intermedio de lectura anticipada se descartan cuando finaliza la aplicación.

En el dominio de publicación/suscripción, JMS identifica dos tipos de consumidor de mensajes: suscriptor de tema no duradero y suscriptor de tema duradero, que se describen en las dos secciones siguientes.

Suscriptores de temas no duraderos

Un suscriptor de tema no duradero sólo recibe aquellos mensajes que se han publicado mientras el suscriptor está activo. Una suscripción no duradera empieza cuando una aplicación crea un suscriptor de tema no duradero y finaliza cuando la aplicación cierra el suscriptor o cuando el suscriptor está fuera del ámbito. Como una extensión en IBM MQ classes for JMS. un suscriptor de tema no duradero también recibe publicaciones retenidas.

Para crear un suscriptor de tema no duradero, una aplicación puede utilizar el método createConsumer() independiente del dominio, y especificar un objeto Topic como destino. O bien, una aplicación puede utilizar el método createSubscriber() específico del dominio, tal como se muestra en el ejemplo siguiente:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

El parámetro `topic` es un objeto Topic que la aplicación ha creado anteriormente.

Suscriptores de temas duraderos

Restricción: Una aplicación no puede crear suscriptores de temas duraderos cuando utiliza una conexión en tiempo real con un intermediario.

Un suscriptor de temas duradero recibe todos los mensajes que se publican durante el ciclo de vida de una suscripción duradera. Estos mensajes incluyen todos aquellos que se publican mientras el suscriptor no está activo. Como una extensión en IBM MQ classes for JMS, un suscriptor de tema duradero también recibe publicaciones retenidas.

Para crear un suscriptor de temas duradero, una aplicación utiliza el método createDurableSubscriber()de un objeto Session, tal como se muestra en el ejemplo siguiente:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

En la llamada createDurableSubscriber(), el primer parámetro es un objeto Topic que la aplicación ha creado anteriormente y el segundo parámetro es un nombre que se utiliza para identificar la suscripción duradera.

La sesión que sirve para crear un suscriptor de temas duradero debe tener asociado un identificador de cliente. El identificador de cliente que está asociado a una sesión es el mismo que el del identificador de cliente de la conexión que se utiliza para crear la sesión. El identificador de cliente se puede especificar estableciendo la propiedad CLIENTID del objeto ConnectionFactory. De forma alternativa, una aplicación puede especificar el identificador de cliente invocando el método setClientID() del objeto Connection.

El nombre que se utiliza para identificar una suscripción duradera sólo debe ser exclusivo en el ámbito del identificador de cliente y, por consiguiente, el identificador de cliente forma parte del identificador completo y exclusivo de una suscripción duradera. Para seguir utilizando una suscripción duradera que se ha creado anteriormente, una aplicación debe crear un suscriptor de temas duradero mediante una sesión con el mismo identificador de cliente que el que está asociado a una suscripción duradera y con el mismo nombre de suscripción.

Una suscripción duradera se inicia cuando una aplicación crea un suscriptor de tema duradero con un identificador de cliente y un nombre de suscripción para el que no existe actualmente ninguna suscripción duradera. Sin embargo, una suscripción duradera no finaliza cuando la aplicación cierra el suscriptor de temas duradero. Para finalizar una suscripción duradera, una aplicación deber invocar el método unsubscribe() de un objeto Session que tenga el mismo identificador de cliente que el que está asociado a la suscripción duradera. El parámetro en la llamada unsubscribe() es el nombre de suscripción que el que figura en el ejemplo siguiente:

```
session.unsubscribe("D_SUB_000001");
```

El ámbito de una suscripción duradera es un gestor de colas. Si una suscripción duradera existe en un gestor de colas y una aplicación conectada a otro gestor de colas crea una suscripción duradera con el mismo identificador de cliente y el mismo nombre de suscripción, las dos suscripciones duraderas son completamente independientes.

Selectores de mensaje

Una aplicación sólo puede especificar que sólo aquellos mensajes que cumplen determinados criterios se devuelvan mediante llamadas receive() sucesivas. Cuando se crea un objeto MessageConsumer, la aplicación puede especificar una expresión Structured Query Language (SQL) que determina qué mensajes se recuperan. Esta expresión SQL se denomina *selector de mensajes*. El selector de mensajes puede contener los nombres de los campos de cabecera de mensaje y las propiedades del mensaje de JMS. Para obtener información sobre cómo crear un identificador de mensajes, consulte “Selectores de mensajes en JMS” en la página 150.

El ejemplo siguiente muestra cómo una aplicación puede seleccionar mensajes de acuerdo con una propiedad definida por el usuario denominada myProp:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

La especificación JMS no permite que una aplicación cambie el selector de mensajes de un consumidor de mensajes. Después de que una aplicación cree un consumidor de mensajes con un selector de mensajes, el selector persiste durante el tiempo de vida de ese consumidor. Si una aplicación necesita más de un selector de mensajes, la aplicación debe crear un consumidor de mensajes para cada selector de mensajes.

Observe que, cuando una aplicación está conectada al gestor de colas de la versión 7, la propiedad MSGSELECTION de la fábrica de conexiones no tiene ningún efecto. Para optimizar el rendimiento, toda la selección de mensajes es realizada por el gestor de colas.

Supresión de publicaciones locales

Una aplicación puede crear un consumidor de mensajes que pasa por alto las publicaciones publicadas en la propia conexión del consumidor. ´Para ello, la aplicación establece el tercer parámetro de una llamada createConsumer() en el valor `true`, tal como se muestra en el siguiente ejemplo:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

En una llamada createDurableSubscriber(), la aplicación hace esto estableciendo el cuarto parámetro en el valor `true`, tal como se muestra en el siguiente ejemplo

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Entrega asíncrona de mensajes

Una aplicación puede recibir mensajes de forma asíncrona mediante el registro de un escucha de mensajes en un consumidor de mensajes. El escucha de mensajes tiene un método denominado onMessage, que se llama asíncronamente cuando está disponible un mensaje adecuado y cuya finalidad es procesar el mensaje. El código siguiente ilustra el mecanismo:

V 9.3.0 JM 3.0 V 9.3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
```

```
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

Una aplicación puede utilizar una sesión para recibir mensajes síncronamente con llamadas receive() o bien para recibir mensajes asíncronamente con escuchas de mensajes, pero no para ambas cosas. Si una aplicación necesita recibir mensajes síncrona y asíncronamente, debe crear sesiones distintas.

Una vez que se ha configurado una sesión para recibir mensajes de forma asíncrona, no se pueden invocar los métodos siguientes en esa sesión o para objetos creados desde esa sesión:

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()
- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- MessageProducer.send(Destination, Message, CompletionListener)
- MessageProducer.send(Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send(Message, CompletionListener)
- MessageProducer.send(Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()

- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

Si se llama a alguno de estos métodos, una excepción JMSException que contiene el mensaje:

```
JMSCC0033: No se permite una llamada de método síncrono cuando se utiliza una sesión de forma
asíncrona: 'nombre de método'
```

se genera.

Recepción de mensajes no entregables

Una aplicación puede recibir un mensaje que no se pueda procesar. Puede haber varias razones por las que el mensaje no se puede procesar, por ejemplo, el mensaje puede tener un formato incorrecto. Esos mensajes se describen como mensajes no entregables y necesitan un manejo especial para evitar que el mensaje se procese repetidamente.

Para obtener detalles sobre cómo manejar mensajes no entregables, consulte “Manejo de mensajes no entregables en IBM MQ classes for JMS” en la página 240.

Personalización de los tamaños de almacenamiento intermedio para que se adapten a los mensajes que se reciben

Cuando una aplicación que no es deJMS recibe un mensaje de IBM MQ , la aplicación debe proporcionar un almacenamiento intermedio de mensajes para que el mensaje se escriba en él. Las aplicaciones JMS no necesitan crear manualmente un almacenamiento intermedio. El IBM MQ classes for JMS crea y dimensiona automáticamente los almacenamientos intermedios de mensajes para que se ajusten a los tamaños de los mensajes que se reciben. Para la mayoría de las aplicaciones, los almacenamientos intermedios gestionados automáticamente proporcionan un equilibrio adecuado de rendimiento y comodidad para el desarrollador de aplicaciones. En determinadas circunstancias, puede ser beneficioso especificar el tamaño inicial del almacenamiento intermedio de mensajes manualmente. El tamaño inicial predeterminado de un almacenamiento intermedio de recepción de IBM MQ JMS es de 4 KB. Si una aplicación siempre va a recibir mensajes de 256 KB de tamaño, es posible que sea preferible configurar el tamaño del almacenamiento intermedio inicial a 256 KB. Esto puede evitar la necesidad de que IBM MQ classes for JMS intente y no reciba el mensaje en un almacenamiento intermedio de 4 KB antes de redimensionarlo a 256 KB y recibirlo correctamente. Para una aplicación conectada al cliente, esto puede evitar la necesidad de un viaje de ida y vuelta de red potencialmente desperdiciado mientras el IBM MQ classes for JMS determina el tamaño de almacenamiento intermedio correcto que se debe utilizar.

El tamaño del almacenamiento intermedio inicial se puede configurar estableciendo la propiedad com.ibm.mq.jmqi.defaultMaxMsgSize Java en el valor elegido, en bytes. Tenga en cuenta que esta propiedad afecta a todas las aplicaciones de IBM MQ JMS que se ejecutan dentro de Java Virtual Machine, por lo tanto, tenga cuidado de no afectar negativamente a otros consumidores de mensajes que reciben mensajes de un tamaño diferente.

El IBM MQ classes for JMS sigue intentando reducir automáticamente el tamaño del almacenamiento intermedio si se reciben varios mensajes más pequeños que el tamaño configurado. De forma predeterminada, esto sucede si se reciben 10 mensajes que son todos más pequeños que el tamaño del almacenamiento intermedio. Por ejemplo, si se reciben 10 mensajes en una fila con un tamaño de 128 KB, el almacenamiento intermedio se reduce de 256 KB a 128 KB. A continuación, se vuelve a aumentar cuando se reciben mensajes más grandes. Es posible configurar el número de mensajes que se deben recibir antes de que se reduzca el tamaño de un almacenamiento intermedio. Por ejemplo, esto puede ser útil si se sabe que la aplicación recibe cinco mensajes grandes seguidos de 10 mensajes más pequeños y, a continuación, otros cinco mensajes grandes. Con los valores predeterminados, el almacenamiento intermedio se reduciría después de que se hubieran recibido los 10 mensajes más pequeños y sería necesario aumentar de nuevo para los mensajes más grandes. La propiedad del sistema Java com.ibm.mq.jmqi.smallMsgBufferReductionThreshold se puede establecer en el número de mensajes que se deben recibir antes de que se reduzca el tamaño del almacenamiento intermedio. En

este ejemplo, se podría establecer en 20 para evitar que 10 mensajes más pequeños reduzcan el tamaño del almacenamiento intermedio.

Las propiedades se pueden establecer independientemente entre sí. Por ejemplo, puede optar por dejar el tamaño del almacenamiento intermedio inicial en su valor predeterminado de 4 KB, pero aumentar el valor de com.ibm.mq.jmqi.smallMsgBufferReductionThreshold , por lo que una vez que el almacenamiento intermedio se aumente en tamaño, permanecerá ese tamaño durante más tiempo.

Si se ven grandes números de códigos de retorno MQRC_TRUNCATED_MSG_FAILED (2080) para las aplicaciones JMS en los registros de estadísticas MQI, esto podría ser una indicación de que se beneficiaría de configurar un tamaño de almacenamiento intermedio inicial más alto para dichas aplicaciones, o de reducir la frecuencia con la que se reducen los tamaños de almacenamiento intermedio. Sin embargo, es importante tener en cuenta que para una aplicación de larga ejecución es probable que sólo vea un número muy pequeño de códigos de retorno MQRC_TRUNCATED_MSG_FAILED. Esto se debe a que normalmente el almacenamiento intermedio se aumenta al tamaño correcto inmediatamente después de que se reciba el primer mensaje grande, y no se reduce en tamaño a menos que se reciba un número de mensajes más pequeños. Por lo tanto, es posible que un gran número de MQRC_TRUNCATED_MSG_FAILED indique otras prácticas de aplicación deficientes como, por ejemplo, conectarse a IBM MQ para recibir sólo uno o dos mensajes antes de desconectarse.

Recuperación de datos de usuario de suscripción

Si los mensajes que una aplicación de IBM MQ classes for JMS está consumiendo de una cola se colocan mediante una suscripción duradera definida administrativamente, la aplicación necesita acceder a la información de datos de usuario que está asociada con la suscripción. Esta información se añade al mensaje como una propiedad.

Cuando se consume un mensaje de una cola que contiene una cabecera RFH2 con la carpeta MQPS, el valor que está asociado con la clave Sud, si existe, se añade como una propiedad Serie al objeto de mensaje JMS devuelto a la aplicación de IBM MQ classes for JMS. Para habilitar la recuperación de esta propiedad del mensaje, se puede utilizar la constante JMS_IBM_SUBSCRIPTION_USER_DATA en la interfaz JmsConstants con el método siguiente para obtener los datos de usuario de suscripción:

- V 9.3.0 JM 3.0 V 9.3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

En el ejemplo siguiente, una suscripción duradera administrativa se define utilizando el mandato MQSC **DEFINE SUB**:

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

Las copias de los mensajes que se publican en la serie de tema PUBLIC se colocan en la cola MY.SUBSCRIPTION.Q. A continuación, los datos de usuario asociados a la suscripción duradera se añaden como una propiedad al mensaje, que se almacena en la carpeta MQPS de la cabecera RFH2 con la clave Sud.

La aplicación de IBM MQ classes for JMS puede llamar a:

V 9.3.0 JM 3.0 V 9.3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0
```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

Se devuelve la serie siguiente:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

Conceptos relacionados

“La cabecera MQRFH2 y JMS” en la página 155

Tareas relacionadas

Definir una suscripción administrativa

Referencia relacionada

DEFINE SUB

Interfaz JmsConstants

Cierre de una aplicación de IBM MQ classes for JMS

Es importante que una aplicación de IBM MQ classes for JMS cierre explícitamente determinados objetos de JMS antes de detenerse. Es posible que no se invoquen métodos finalizadores, por lo que no puede depender de ellos para liberar recursos. No permita que una aplicación termine con un rastreo comprimido activo.

La recogida de basura por sí sola no puede liberar puntualmente todos los recursos de IBM MQ classes for JMS y IBM MQ, especialmente si una aplicación crea muchos objetos temporales de JMS a nivel de sesión o inferior. Por lo tanto, es importante que una aplicación cierre un objeto Connection, Session, MessageConsumer o MessageProducer, cuando ya no sea necesario.

Cuando una aplicación cierra una conexión, se produce una retrotracción implícita para todas las sesiones de transacción de la conexión. Para asegurarse de que se hayan confirmado los cambios realizados por la aplicación, cierre la conexión explícitamente antes de cerrar la aplicación.

No utilice métodos finalizadores en una aplicación para cerrar objetos de JMS. Puesto que es posible que no se invoquen finalizadores, puede que no se liberen recursos. Cuando se cierra una conexión, cierra todas las sesiones que se han creado a partir de la sesión. Similarmente, los objetos MessageConsumers y MessageProducers creados desde una sesión se cierran cuando se cierra la sesión. Pero considere la posibilidad de cerrar Sessions, MessageConsumers y MessageProducers explícitamente para asegurar la liberación puntual de los recursos.

Si se activa la compresión de rastreo, System.Halt() concluye y es probable que las terminaciones anómalas y no controladas de la JVM produzcan un archivo de rastreo dañado. Cuando sea posible, desactive el recurso de rastreo cuando haya recopilado la información de rastreo que necesite. Si está rastreando una aplicación hasta una terminación anómala, utilice la salida de rastreo no comprimida.

Nota: Para desconectar de un gestor de colas, una aplicación JMS invoca el método close() en el objeto de conexión.

Manejo de mensajes no entregables en IBM MQ classes for JMS

Un mensaje con formato incorrecto es uno que no puede ser procesado por una aplicación receptora. Si se entrega un mensaje con formato incorrecto a una aplicación y se retrotrae un número especificado de veces, las IBM MQ classes for JMS pueden moverlo a una cola de retirada.

Un mensaje con formato incorrecto es un mensaje que no puede ser procesado por una aplicación receptora. El mensaje podría tener un tipo inesperado o contener información que no puede manejar la lógica de la aplicación. Si se entrega un mensaje con formato incorrecto a una aplicación, la aplicación no podrá procesarlo y lo retrotraerá hasta su cola de procedencia. De forma predeterminada, las IBM MQ classes for JMS volverán entregar el mensaje a la aplicación repetidamente. Esto puede dar como resultado que la aplicación se atasque en un bucle de forma continuada intentando procesar el mensaje con formato incorrecto y retrotrayéndolo.

Para evitar que esto suceda, las IBM MQ classes for JMS pueden detectar mensajes con formato incorrecto y trasladarlos a un destino alternativo. Para ello, las IBM MQ classes for JMS utilizan las propiedades siguientes:

- El valor del campo BackoutCount dentro de MQMD del mensaje que se ha detectado.
- Los atributos de cola IBM MQ **BOTHRESH** (umbral de restitución) y **BOQNAME** (cola para volver a poner en cola de restitución) para la cola de entrada que contiene el mensaje.

Siempre que una aplicación retrotrae un mensaje, el gestor de colas aumenta de forma automática el valor del campo BackoutCount para el mensaje.

Cuando las IBM MQ classes for JMS detectan un mensaje que tiene un valor de BackoutCount mayor que cero, comparan el valor de BackoutCount con el valor del atributo **BOTHRESH**.

- Si el valor de BackoutCount es menor que el valor del atributo **BOTHRESH**, las IBM MQ classes for JMS lo entregan a la aplicación para su proceso.
- Sin embargo, si el valor de BackoutCount es mayor o igual que **BOTHRESH**, se considera que el mensaje es un mensaje con formato incorrecto. En esta situación, las IBM MQ classes for JMS trasladan el mensaje a la cola especificada por el atributo **BOQNAME**. Si el mensaje no se puede colocar en la cola de restitución, se mueve a la cola de mensajes no entregados del gestor de colas o se descarta, en función de las opciones de informe del mensaje.

Nota:

- Si el atributo **BOTHRESH** se deja en su valor predeterminado de `0`, el manejo de mensajes con formato incorrecto se inhabilita. Esto significa que los mensajes con formato incorrecto se vuelven a colocar en la cola de entrada.
- La otra cosa que hay que tener en cuenta es que las IBM MQ classes for JMS consultan los atributos **BOTHRESH** y **BOQNAME** para la cola la primera vez que detectan un mensaje que tiene un valor de BackoutCount mayor que cero. Los valores de estos atributos se almacenan en la memoria caché y se utilizan siempre que las IBM MQ classes for JMS encuentran un mensaje que tiene un valor de BackoutCount mayor que cero.

Configuración del sistema para realizar el manejo de mensajes con formato incorrecto

La cola que utilizan las IBM MQ classes for JMS al consultar los atributos **BOTHRESH** y **BOQNAME** depende del estilo de mensajería que se está realizando.

- Para la mensajería punto a punto, esta es la cola local subyacente. Esto es importante cuando una aplicación JMS consume mensajes de colas de alias o de colas de clúster.
- Para la mensajería de publicación/suscripción, se crea una cola gestionada para contener los mensajes para una aplicación. Las IBM MQ classes for JMS consultan la cola gestionada para determinar los valores para los atributos **BOTHRESH** y **BOQNAME**.

  La cola gestionada se crea a partir de una cola de modelo asociada al objeto de tema al que está suscrita la aplicación y hereda los valores de los atributos **BOTHRESH** y **BOQNAME** de la cola de modelo. La cola de modelo que se utiliza depende de si la aplicación receptora ha extraído una suscripción duradera o no duradera:

  – La cola de modelo utilizada para las suscripciones duraderas se especifica mediante el atributo **MDURMDL** del tema. El valor predeterminado de este atributo es `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Para las suscripciones no duraderas, la cola de modelo que se utiliza se especifica mediante el atributo **MNDURMDL**. El valor predeterminado del atributo **MNDURMDL** es `SYSTEM.NDURABLE.MODEL.QUEUE`.

Al consultar los atributos **BOTHRESH** y **BOQNAME**, las IBM MQ classes for JMS:

- Abren la cola local, o la cola de destino para una cola alias.
- Consultan los atributos **BOTHRESH** y **BOQNAME**.
- Cierran la cola local, o la cola de destino para una cola alias.

Las opciones de abrir que se utilizan al abrir la cola local, o la cola de destino para una cola alias, dependen de la versión de las IBM MQ classes for JMS que se están utilizando:

- Para IBM MQ classes for JMS en IBM MQ 9.1.0 Fix Pack 1 y anteriores, o IBM MQ 9.1.1, si la cola local, o la cola de destino para una cola alias, es una cola de clúster, IBM MQ classes for JMS abra la cola con las opciones `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` y `MQOO_FAIL_IF_QUIESCING` . Esto significa que el usuario que ejecuta la aplicación receptora debe tener acceso de consulta y obtención en la instancia local de la cola del clúster.

Las IBM MQ classes for JMS abren todos los demás tipos de cola local con las opciones para abrir MQOO_INQUIRE y MQOO_FAIL_IF_QUIESCING. Para que las IBM MQ classes for JMS puedan consultar los valores de los atributos, el usuario que ejecuta la aplicación receptora debe tener acceso de consulta sobre la cola local.

- Cuando se utiliza IBM MQ classes for JMS en IBM MQ 9.1.0 Fix Pack 2 y posteriores, o para IBM MQ 9.1.2 y posteriores, el usuario que ejecuta la aplicación receptora debe tener acceso de consulta en la cola local, independientemente del tipo de cola.

Para mover mensajes con formato incorrecto a una cola de reposición en cola de restitución o a la cola de mensajes no entregados del gestor de colas, debe otorgar al usuario que ejecuta la aplicación las autorizaciones put y passall .

Proceso de mensajes con formato incorrecto para aplicaciones síncronas

Si una aplicación recibe mensajes de forma síncrona, llamando uno de los métodos siguientes, las IBM MQ classes for JMS vuelven a colocar en cola un mensaje con formato incorrecto dentro de la unidad de trabajo que estaba activa cuando la aplicación intentó obtener el mensaje:

- JMSConsumer.receive()
- JMSConsumer.receive(long timeout)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive()
- MessageConsumer.receive(long timeout)
- MessageConsumer.receiveNoWait()
- QueueReceiver.receive()
- QueueReceiver.receive(long timeout)
- QueueReceiver.receiveNoWait()
- TopicSubscriber.receive()
- TopicSubscriber.receive(long timeout)
- TopicSubscriber.receiveNoWait()

Esto significa que si la aplicación está utilizando un contexto o una sesión de JMS transaccionado, el traslado del mensaje a la cola de retirada no se confirma hasta que se confirma la transacción.

Si el atributo **BOTHRESH** está establecido en un valor distinto a cero, el atributo **BOQNAME** también se debe establecer. Si **BOTHRESH** está establecido en un valor mayor que cero, y no se ha establecido **BOQNAME**, el comportamiento se determina mediante las opciones de informe del mensaje:

- Si el mensaje tiene la opción de informe MQRO_DISCARD_MSG establecida, el mensaje se descarta.
- Si el mensaje tiene la opción de informe MQRO_DEAD_LETTER_Q especificada, el IBM MQ classes for JMS intenta mover el mensaje a la cola de mensajes no entregados del gestor de colas.
- Si el mensaje no tiene MQRO_DISCARD_MSG o MQRO_DEAD_LETTER_Q establecido, las IBM MQ classes for JMS intentan colocar el mensaje en la cola de mensajes no entregados para el gestor de colas.

En el supuesto de que el intento de colocar el mensaje en la cola de mensajes no entregados falle por algún motivo, lo que sucede en el mensaje está determinado por si la aplicación receptora está utilizando un contexto o una sesión de JMS transaccionado o no transaccionado:

- Si la aplicación receptora está utilizando un contexto o una sesión de JMS transaccionado, y la transacción se confirma, el mensaje se descarta.
- Si la aplicación receptora está utilizando un contexto o una sesión de JMS transaccionado, y retrotrae la transacción, el mensaje se devuelve a la cola de entrada.

- Si la aplicación receptora ha creado un contexto o una sesión de JMS no transaccionado, el mensaje se descarta.

Proceso de mensajes con formato incorrecto para aplicaciones asíncronas

Si una aplicación está recibiendo mensajes de forma asíncrona a través de MessageListener, las IBM MQ classes for JMS vuelven a colocar en cola los mensajes con formato incorrecto sin afectar a la entrega de mensajes. El proceso para volver a poner en cola se realiza fuera de cualquier unidad de trabajo asociada a la entrega real de mensajes en la aplicación.

Si **BOTHRESH** está establecido en un valor mayor que cero, y no se ha establecido **BOQNAME**, el comportamiento se determina mediante las opciones de informe del mensaje:

- Si el mensaje tiene la opción de informe `MQRO_DISCARD_MSG` establecida, el mensaje se descarta.
- Si el mensaje tiene la opción de informe `MQRO_DEAD_LETTER_Q` especificada, el IBM MQ classes for JMS intenta mover el mensaje a la cola de mensajes no entregados del gestor de colas.
- Si el mensaje no tiene `MQRO_DISCARD_MSG` o `MQRO_DEAD_LETTER_Q` establecido, las IBM MQ classes for JMS intentan colocar el mensaje en la cola de mensajes no entregados para el gestor de colas.

Si el intento de colocar el mensaje en la cola de mensajes no entregados falla por algún motivo, las IBM MQ classes for JMS devuelven el mensaje a la cola de entrada.

Si desea más información sobre cómo las especificaciones de activación y ConnectionConsumers manejan los mensajes con formato incorrecto, consulte Eliminación de mensajes de la cola en ASF.

Qué sucede en un mensaje cuando se traslada a la cola de retirada

Cuando un mensaje con formato incorrecto se vuelve a poner en la cola para volver a poner en cola de retirada, las IBM MQ classes for JMS le añaden una cabecera RFH2 (si todavía no tenía ninguna) y actualizan alguno de los campos del descriptor de mensaje (MQMD).

Si el mensaje con formato incorrecto contiene una cabecera RFH2 (por ejemplo, porque era un mensaje JMS), las IBM MQ classes for JMS cambian los campos siguientes dentro de MQMD cuando se traslada el mensaje a la cola para volver a poner en cola de retirada:

- El campo BackoutCount se restablece en cero.
- El campo de caducidad del mensaje se actualiza para reflejar la caducidad restante en el momento cuando la aplicación JMS recibió el mensaje con formato incorrecto.

Si el mensaje con formato incorrecto no contiene una cabecera RFH2, las IBM MQ classes for JMS añaden una y actualizan los campos siguientes en MQMD como parte del proceso de restitución:

- El campo BackoutCount se restablece en cero.
- El campo de caducidad del mensaje se actualiza para reflejar la caducidad restante en el momento cuando la aplicación JMS recibió el mensaje con formato incorrecto.
- El campo de formato del mensaje se cambia a MQHRF2.
- El campo CCSID se cambia para que sea 1208.
- El campo de codificación se modifica para que sea 273.

Además de esto, los campos CCSID y de codificación del mensaje con formato incorrecto se copian en los campos CCSID y de codificación de la cabecera RFH2, para asegurarse de que el encadenamiento de la cabecera del mensaje en la cola para volver a poner en cola de retirada es correcto.

Conceptos relacionados

“Manejo de mensajes dañados en ASF” en la página 346
Dentro de ASF (Application Server Facilities), los mensajes no entregables se manejan de forma ligeramente diferente a como se manejan en otros lugares de IBM MQ classes for JMS.

Excepciones en IBM MQ classes for JMS

Una aplicación IBM MQ classes for JMS debe manejar las excepciones que emiten las llamadas de API de JMS o que se entregan a un manejador de excepciones.

IBM MQ classes for JMS notifica problemas en tiempo de ejecución emitiendo excepciones. El tipo de excepciones que se emiten, y la forma en que se deben manejar estas excepciones, depende de la versión de la especificación JMS que utiliza la aplicación:

- Los métodos en las interfaces definidas en JMS 1.1 y anteriores emiten excepciones comprobadas. La clase base de estas excepciones es `JMSException`. Para obtener más información sobre cómo manejar las excepciones comprobadas, consulte "Manejo de excepciones comprobadas" en la página 244.
- Los métodos de las interfaces añadidas en JMS 2.0 generan excepciones no comprobadas. La clase base para estas excepciones es `JMSRuntimeException`. Para obtener más información sobre cómo manejar excepciones no comprobadas, consulte "Manejo de excepciones no comprobadas" en la página 247.

También puede registrar un `ExceptionListener` con un JMS `Connection` o un `JMSContext`. A continuación, las clases de MQ para JMS notifican a `ExceptionListener` si se detecta un problema con una conexión con el gestor de colas, o si se produce un problema al intentar entregar un mensaje de forma asíncrona. Para obtener más información, consulte "ExceptionListeners" en la página 250.

Conceptos relacionados

Clases IBM MQ para JMS

Referencia relacionada

ASYNCEXCEPTION

Manejo de excepciones comprobadas

Los métodos en las interfaces definidas en JMS 1.1 o anteriores emiten excepciones comprobadas. La clase base para estas excepciones es `JMSException`. Por lo tanto, la captura de `JMSExceptions` proporciona una forma genérica de manejar estos tipos de excepciones.

Cada `JMSException` encapsula la información siguiente:

- Un mensaje de excepción específico del proveedor, que la aplicación puede obtener llamando al método `Throwable.getMessage()`.
- Un código de error específico del proveedor, que la aplicación puede obtener llamando al método `JMSException.getErrorCode()`.
- Una excepción enlazada. Una excepción generada por una llamada de API JMS 1.1 suele ser el resultado de un problema de nivel inferior notificado por otra excepción que está enlazada a esta excepción. La aplicación puede obtener una excepción enlazada llamando al método `JMSException.getLinkedException()` o al método `Throwable.getCause()`.

Cuando se utiliza la API JMS 1.1 , la mayoría de las excepciones generadas por IBM MQ classes for JMS son instancias de subclases de `JMSException`. Estas subclases implementan la interfaz `com.ibm.msg.client.jms.JmsExceptionDetail` , que proporciona la siguiente información adicional:

- Una explicación del mensaje de excepción. La aplicación puede obtener este mensaje llamando al método `JmsExceptionDetail.getExplanation()`.
- Una respuesta de usuario recomendada a la excepción. La aplicación puede obtener este mensaje llamando al método `JmsExceptionDetail.getUserAction()`.
- Las claves para las inserciones de mensajes en el mensaje de excepción. La aplicación puede obtener un iterador para todas las claves llamando al método `JmsExceptionDetail.getKeys()`.
- Las inserciones de mensajes en el mensaje de excepción. Por ejemplo, una inserción de mensaje puede ser el nombre de la cola que ha causado la excepción y puede ser útil para que la aplicación acceda a dicho nombre. La aplicación puede obtener la inserción de mensaje correspondiente a una clave especificada llamando al método `JmsExceptionDetail.getValue()`.

Todos los métodos de la interfaz `JmsExceptionDetail` devuelven un valor nulo si no hay detalles disponibles.

Por ejemplo, si una aplicación intenta crear un productor de mensajes para una cola IBM MQ que no existe, se genera una excepción con la siguiente información:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

La excepción que se genera, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, es una subclase de la clase siguiente e implementa la interfaz `com.ibm.msg.client.jms.JmsExceptionDetail` .

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Excepciones enlazadas

Una excepción enlazada proporciona más información sobre un problema de tiempo de ejecución. Por lo tanto, para cada `JMSException` que se genera, una aplicación debe comprobar la excepción enlazada.

La propia excepción enlazada puede tener otra excepción enlazada y, por lo tanto, las excepciones enlazadas forman una cadena que conduce de nuevo al problema subyacente original. Una excepción enlazada se implementa utilizando el mecanismo de excepción encadenada de la clase `java.lang.Throwable` y la aplicación puede obtener una excepción enlazada llamando al método `Throwable.getCause()` . Para un `JMSException`, el método `getLinkedException()` delega en el método `Throwable.getCause()` .

Por ejemplo, si una aplicación especifica un número de puerto incorrecto al conectarse a un gestor de colas, las excepciones forman la cadena siguiente:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Normalmente, cada excepción de una cadena se genera a partir de una capa diferente del código. Por ejemplo, las capas siguientes generan las excepciones de la cadena precedente:

- La primera excepción, una instancia de una subclase de `JMSException`, la genera la capa común en IBM MQ classes for JMS.
- El proveedor de mensajería de IBM MQ genera la siguiente excepción, una instancia de `com.ibm.mq.MQException`.
- Las dos excepciones siguientes, ambas de las cuales son instancias de `com.ibm.mq.jmqi.JmqiException`, las emite la JMQI (Java Message Queueing Interface). El JMQI es el componente que utiliza el IBM MQ classes for JMS para comunicarse con un gestor de colas.
- La excepción final, una instancia de `java.net.ConnectionException`, la emite la biblioteca de clases Java .

Para obtener más información sobre la arquitectura en capas de IBM MQ classes for JMS, consulte Arquitectura de clases de IBM MQ para JMS.

Puede codificar la aplicación para que itere a través de esta cadena para extraer toda la información adecuada, tal como se muestra en el ejemplo siguiente:

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
```

```
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Tenga en cuenta que la aplicación siempre debe comprobar el tipo de cada excepción de una cadena porque el tipo de excepción puede variar y las excepciones de distintos tipos encapsulan información diferente.

Obtención de información específica de IBM MQ sobre un problema

Las instancias de `com.ibm.mq.MQException` y `com.ibm.mq.jmqi.JmqiException` encapsulan IBM MQ información específica sobre un problema.

Un `MQException` encapsula la siguiente información:

- Un código de terminación, que la aplicación puede obtener llamando al método `getCompCode()` .
- Un código de razón, que la aplicación puede obtener llamando al método `getReason()` .

Para obtener ejemplos de cómo utilizar estos métodos, consulte el código de ejemplo en excepciones enlazadas.

Un `JmqiException` también encapsula un código de terminación y un código de razón. Además de esto, un `JmqiException` contiene la información en un mensaje AMQ *nnnn* o CSQ *nnnn* , si hay uno asociado con la excepción. La aplicación puede obtener los diversos componentes de este mensaje llamando a los métodos siguientes:

- El método `getWmqMsgExplanation()` devuelve la explicación del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgSeverity()` devuelve la gravedad del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgSummary()` devuelve el resumen del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgUserResponse()` devuelve la respuesta del usuario asociada con el mensaje AMQ *nnnn* o CSQ *nnnn* .

Manejo de excepciones no comprobadas
Los métodos de las interfaces definidas en JMS 2.0 generan excepciones no comprobadas.
La clase base para estas excepciones es `JMSRuntimeException`. Por lo tanto, la captura de `JMSRuntimeExceptions` proporciona una forma genérica de manejar estos tipos de excepciones.

Cada `JMSRuntimeException` encapsula la información siguiente:

- Un mensaje de excepción específico del proveedor, que la aplicación puede obtener llamando al método `JMSRuntimeException.getMessage()` .
- Un código de error específico del proveedor, que la aplicación puede obtener llamando al método `JMSRuntimeException.getErrorCode()` .
- Una excepción enlazada. Una excepción generada por una llamada de API JMS 2.0 suele ser el resultado de un problema de nivel inferior notificado por otra excepción enlazada a esta excepción. La aplicación puede obtener una excepción enlazada llamando al método `JMSRuntimeException.getCause()` .

Cuando llama a métodos en las interfaces proporcionadas por la API JMS 2.0 , la mayoría de las excepciones generadas por IBM MQ classes for JMS son instancias de subclases de `JMSRuntimeException`. Estas subclases implementan la interfaz

`com.ibm.msg.client.jms.JmsExceptionDetail` , que proporciona la siguiente información adicional:

- Una explicación del mensaje de excepción. La aplicación puede obtener este mensaje llamando al método `JmsExceptionDetail.getExplanation()` .
- Una respuesta de usuario recomendada a la excepción. La aplicación puede obtener este mensaje llamando al método `JmsExceptionDetail.getUserAction()` .
- Las claves para las inserciones de mensajes en el mensaje de excepción. La aplicación puede obtener un iterador para todas las claves llamando al método `JmsExceptionDetail.getKeys()` .
- Las inserciones de mensajes en el mensaje de excepción. Por ejemplo, una inserción de mensaje puede ser el nombre de la cola que ha causado la excepción y puede ser útil para que la aplicación acceda a dicho nombre. La aplicación puede obtener la inserción de mensaje correspondiente a una clave especificada llamando al método `JmsExceptionDetail.getValue()` .

Todos los métodos de la interfaz `JmsExceptionDetail` devuelven un valor nulo si no hay detalles disponibles.

Por ejemplo, si una aplicación intenta crear un `JMSProducer` para una cola IBM MQ que no existe, se genera una excepción con la siguiente información:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

La excepción que se genera, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, es una subclase de la clase siguiente e implementa la interfaz `com.ibm.msg.client.jms.JmsExceptionDetail` .

- V 9.3.0 JM 3.0 V 9.3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Excepciones encadenadas

Normalmente, las excepciones se deben a otras excepciones. Por lo tanto, para cada `JMSRuntimeException` que se genera, la aplicación debe comprobar la excepción enlazada.

La causa de `JMSRuntimeException` puede ser otra excepción. Estas excepciones forman una cadena que conduce de nuevo al problema subyacente original. La causa de una excepción se implementa utilizando el mecanismo de excepción encadenada de la clase `java.lang.Throwable` y la aplicación puede obtener una excepción enlazada llamando al método `Throwable.getCause()` .

Por ejemplo, si una aplicación especifica un número de puerto incorrecto al conectarse a un gestor de colas, las excepciones forman la cadena siguiente:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

Normalmente, cada excepción de una cadena se genera a partir de una capa diferente del código. Por ejemplo, las capas siguientes generan las excepciones de la cadena precedente:

- La primera excepción, una instancia de una subclase de `JMSRuntimeException`, la genera la capa común en IBM MQ classes for JMS.
- El proveedor de mensajería de IBM MQ genera la siguiente excepción, una instancia de `com.ibm.mq.MQException`.
- Las dos excepciones siguientes, ambas de las cuales son instancias de `com.ibm.mq.jmqi.JmqiException`, las emite la JMQI (Java Message Queueing Interface). El JMQI es el componente que utiliza el IBM MQ classes for JMS para comunicarse con un gestor de colas.
- La excepción final, una instancia de `java.net.ConnectionException`, la emite la biblioteca de clases Java .

Para obtener más información sobre la arquitectura en capas de IBM MQ classes for JMS, consulte Arquitectura de clases de IBM MQ para JMS.

Puede codificar la aplicación para que itere a través de esta cadena para extraer toda la información adecuada, tal como se muestra en el ejemplo siguiente:

V 9.3.0 JM 3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
```

```
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Tenga en cuenta que la aplicación siempre debe comprobar el tipo de cada excepción de una cadena porque el tipo de excepción puede variar y las excepciones de distintos tipos encapsulan información diferente.

Obtención de información específica de IBM MQ sobre un problema

Las instancias de `com.ibm.mq.MQException` y `com.ibm.mq.jmqi.JmqiException` encapsulan IBM MQ información específica sobre un problema.

Un `MQException` encapsula la siguiente información:

- Un código de terminación, que la aplicación puede obtener llamando al método `getCompCode()` .
- Un código de razón, que la aplicación puede obtener llamando al método `getReason()` .

Para obtener ejemplos de cómo utilizar estos métodos, consulte el código de ejemplo en excepciones encadenadas.

Un `JmqiException` también encapsula un código de terminación y un código de razón. Además de esto, un `JmqiException` contiene la información en un mensaje AMQ *nnnn* o CSQ *nnnn* , si hay uno asociado con la excepción. La aplicación puede obtener los diversos componentes de este mensaje llamando a los métodos siguientes:

- El método `getWmqMsgExplanation()` devuelve la explicación del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgSeverity()` devuelve la gravedad del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgSummary()` devuelve el resumen del mensaje AMQ *nnnn* o CSQ *nnnn* .
- El método `getWmqMsgUserResponse()` devuelve la respuesta del usuario asociada con el mensaje AMQ *nnnn* o CSQ *nnnn* .

ExceptionListeners
Los objetos JMS `Connection` y `JMSContext` tienen una conexión asociada con un gestor de colas. La aplicación puede registrar un `ExceptionListener` con un JMS `Connection` o `JMSContext`. Si se produce un problema que hace que la conexión asociada con el `Connection` o `JMSContext` no se pueda utilizar, el IBM MQ classes for JMS entrega una excepción al `ExceptionListener` llamando a su método `onException()` . A continuación, la aplicación tiene la oportunidad de restablecer la conexión.

IBM MQ classes for JMS también puede entregar una excepción al escucha de excepción si se produce un problema al intentar entregar un mensaje de forma asíncrona.

Escuchas de excepciones

A partir de IBM MQ 8.0.0 Fix Pack 2, para mantener el comportamiento de las aplicaciones JMS actuales que configuran un JMS `MessageListener` y un JMS `ExceptionListener`, y para asegurarse de que los IBM MQ classes for JMS son coherentes con la especificación JMS , el valor predeterminado para la propiedad ASYNCEXCEPTION de `ConnectionFactory` se cambia a ASYNC_EXCEPTIONS_CONNECTIONBROKEN. Como resultado, sólo las excepciones que corresponden a códigos de error de conexión rotos se entregan al `ExceptionListener` de una aplicación.

El APAR IT14820, incluido a partir de IBM MQ 9.0.0 Fix Pack 1, actualiza IBM MQ classes for JMS por lo que:

- Un `ExceptionListener` registrado por una aplicación se invoca para cualquier excepción de conexión interrumpida, independientemente de si la aplicación utiliza consumidores de mensajes síncronos o asíncronos.
- Las excepciones interrumpidas sin conexión (por ejemplo, MQRC_GET_inhibiTED) que surgen durante la entrega de mensajes se entregan al `ExceptionListener` de una aplicación cuando la aplicación utiliza consumidores de mensajes asíncronos y la JMS ConnectionFactory que utiliza la aplicación tiene la propiedad ASYNC_EXCEPTION establecida en el valor ASYNC_EXCEPTIONS_ALL.

Nota: Un `ExceptionListener` sólo se invoca una vez para una excepción de conexión interrumpida, incluso si se interrumpe dos conexiones TCP/IP (una utilizada por una conexión JMS y otra utilizada por una sesión JMS).

Para cualquier otro tipo de problema, la llamada a la API JMS actual genera una excepción. El tipo de excepción que se genera depende de la versión de la API de JMS que la aplicación está utilizando:

- Si la aplicación utiliza las interfaces proporcionadas por la especificación JMS 1.1 , la excepción es un `JMSException`. Para obtener más información sobre cómo manejar estas excepciones, consulte “Manejo de excepciones comprobadas” en la página 244.
- Si la aplicación utiliza interfaces JMS 2.0 , la excepción es un `JMSRuntimeException`. Para obtener más información sobre cómo manejar estas excepciones, consulte “Manejo de excepciones no comprobadas” en la página 247.

Si una aplicación no registra un escucha de excepciones con un `Connection` o `JMSContext`, las excepciones que se entregarían al escucha de excepciones se graban en el registro de IBM MQ classes for JMS .

Acceso a la funcionalidad de IBM MQ desde una aplicación IBM MQ classes for JMS

IBM MQ classes for JMS proporciona recursos para explotar una serie de funciones de IBM MQ.

Atención: Estas funciones están fuera de la especificación JMS o, en determinados casos, incumplen la especificación JMS. Si la utiliza, es poco probable que la aplicación sea compatible con otros proveedores JMS. La funcionalidad que incumple JMS está etiquetada con un aviso de atención.

Lectura y escritura del descriptor de mensaje desde una aplicación de IBM MQ classes for JMS

Puede controlar la capacidad para acceder al descriptor de mensaje (MQMD) estableciendo propiedades en un destino y un mensaje.

Algunas aplicaciones de IBM MQ necesitan que se definan valores específicos en la cabecera MQMD de los mensajes que se les envían. IBM MQ classes for JMS proporciona atributos de mensaje que permiten que las aplicaciones de JMS establezcan campos de MQMD y, por tanto, permiten que las aplicaciones de JMS "controlen" aplicaciones de IBM MQ.

Debe establecer la propiedad de objeto de destino WMQ_MQMD_WRITE_ENABLED en true para que el establecimiento de propiedades de MQMD sea efectivo. A continuación, puede utilizar los métodos de establecimiento de propiedades del mensaje (por ejemplo, setStringProperty) para asignar valores a los

campos de MQMD. Se representan todos los campos de MQMD, excepto StrucId y Version. BackoutCount se puede leer, pero no se puede escribir en él.

Este ejemplo provoca que un mensaje se coloque en una cola o un tema con MQMD.UserIdentifier establecido en"JoeBloggs".

```
// 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
// ...
```

Es necesario establecer WMQ_MQMD_MESSAGE_CONTEXT antes de hacer lo propio con JMS_IBM_MQMD_UserIdentifier. Para obtener más información sobre la utilización de WMQ_MQMD_MESSAGE_CONTEXT, consulte el apartado “Propiedades de objeto de mensaje de JMS” en la página 254.

De manera parecida, puede extraer el contenido de los campos MQMD estableciendo WMQ_MQMD_READ_ENABLED en true antes de recibir un mensaje y, a continuación, utilizar los métodos get del mensaje, como getStringProperty. Las propiedades recibidas son de solo lectura.

Este ejemplo tiene como resultado que el campo *valor* conserva el valor del campo MQMD.ApplIdentityData de un mensaje recibido desde una cola o un tema.

```
// 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");
```

Propiedades del objeto Destination de JMS
Dos propiedades del objeto Destination controlan el acceso a MQMD desde JMS, y una tercera propiedad controla el contexto de mensaje.

Tabla 36. Nombres y descripciones de propiedades

Propiedad	**Formato abreviado**	**Descripción**
WMQ_MQMD_WRITE_ENABLED	MDW	Determina si una aplicación de JMS puede establecer los valores de campos de MQMD
WMQ_MQMD_READ_ENABLED	MDR	Determina si una aplicación de JMS puede extraer los valores de campos de MQMD

Tabla 36. Nombres y descripciones de propiedades (continuación)

Propiedad	Formato abreviado	Descripción
WMQ_MQMD_MESSAGE_ CONTEXT	MDCTX	Determina el nivel de contexto de mensaje que debe ser establecido por la aplicación de JMS. La aplicación se debe ejecutar con la autorización de contexto apropiada para que esta propiedad entre en vigor

Tabla 37. Nombres, valores y métodos set de propiedades

Propiedad	Valores válidos en herramienta de administración (valores predeterminados en negrita)	Valores válidos en programas	Método set
WMQ_MQMD_WRITE _HABILITADO	• **NO** Todas las propiedades JMS_IBM_MQMD* se ignoran y sus valores no se copian en la estructura MQMD subyacente. • SÍ Se procesan las propiedades JMS_IBM_MQMD*. Sus valores se copian en la estructura MQMD subyacente.	• **False** • Sí	setMQMDWriteEnabled
WMQ_MQMD_READ _HABILITADO	• **NO** Al enviar mensajes, las propiedades JMS_IBM_MQMD* en un mensaje enviado no se actualizan para reflejar los valores de campo actualizados en el MQMD. Al recibir mensajes, ninguna de las propiedades JMS_IBM_MQMD* está disponible para un mensaje recibido, incluso si el emisor ha establecido todas o algunas de ellas. • SÍ Al enviar mensajes, todas las propiedades JMS_IBM_MQMD* para un mensaje enviado se actualizan para reflejar los valores de campo actualizados contenidos en MQMD, incluidas las propiedades que el emisor no estableció explícitamente. Cuando se reciben mensajes, todas las propiedades JMS_IBM_MQMD* están disponibles para un mensaje recibido, incluidas las propiedades que el emisor no ha establecido explícitamente.	• **False** • Sí	setMQMDReadEnabled

Tabla 37. Nombres, valores y métodos set de propiedades (continuación)			
Propiedad	**Valores válidos en herramienta de administración (valores predeterminados en negrita)**	**Valores válidos en programas**	**Método set**
WMQ_MQMD _CONTEXTO_MENSAJE	• **DEFAULT** La llamada de API MQOPEN y la estructura MQPMO no especifican opciones de contexto de mensaje explícitas • SET_IDENTITY_CONTEXT La llamada de API MQOPEN especifica la opción de contexto de mensaje MQOO_SET_IDENTITY_CONTEXT y la estructura MQPMO especifica MQPMO_SET_IDENTITY_CONTEXT • SET_ALL_CONTEXT La llamada de API MQOPEN especifica la opción de contexto de mensaje MQOO_SET_ALL_CONTEXT y la estructura MQPMO especifica MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

Propiedades de objeto de mensaje de JMS
Las propiedades de objeto de mensaje con el prefijo JMS_IBM_MQMD permiten establecer o leer el campo de MQMD correspondiente.

Envío de mensajes

Están representados todos los campos MQMD excepto StrucId y Version. Estas propiedades solo hacen referencia a los campos MQMD; donde una propiedad se produce tanto en MQMD como en la cabeceraMQRFH2, la versión en MQRFH2 no se establece ni se extrae.

Se puede establecer cualquiera de estas propiedades, excepto JMS_IBM_MQMD_BackoutCount. Se ignora cualquier valor establecido para JMS_IBM_MQMD_BackoutCount.

Si una propiedad tiene una longitud máxima y proporciona un valor que es demasiado largo, el valor se trunca.

Para algunas propiedades, también debe establecer la propiedad WMQ_MQMD_MESSAGE_CONTEXT en el objeto Destination. La aplicación debe estar en ejecución con la autoridad de contexto adecuada para que esta propiedad tenga efecto. Si no establece WMQ_MQMD_MESSAGE_CONTEXT en un valor adecuado, se hace caso omiso del valor de la propiedad. Si establece WMQ_MQMD_MESSAGE_CONTEXT en un valor adecuado, pero no tiene autoridad de contexto suficiente para el gestor de colas, se emite una excepción JMSException. Las propiedades que necesitan valores específicos de WMQ_MQMD_MESSAGE_CONTEXT son las siguientes.

Las siguientes propiedades necesitan que WMQ_MQMD_MESSAGE_CONTEXT se establezca en WMQ_MDCTX_SET_IDENTITY_CONTEXT o WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

Las siguientes propiedades necesitan que WMQ_MQMD_MESSAGE_CONTEXT se establezca en WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

Recepción de mensajes

Todas estas propiedades están disponibles en un mensaje recibido si la propiedad WMQ_MQMD_READ_ENABLED se establece en true, independientemente de las propiedades reales que la aplicación productora haya establecido. Una aplicación no puede modificar las propiedades de un mensaje recibido a menos que primero se borren todas las propiedades, de acuerdo con la especificación JMS. El mensaje recibido se puede enviar sin modificar las propiedades.

Atención: Si la aplicación recibe un mensaje desde un destino con la propiedad WMQ_MQMD_READ_ENABLED establecida en true y lo reenvía a un destino con la propiedad WMQ_MQMD_WRITE_ENABLED establecida en true, esto tiene como resultado que todos los valores de los campos MQMD del mensaje recibido se copien en el mensaje reenviado.

Tabla de propiedades

En esta tabla se listan las propiedades del objeto de mensaje que representan los campos MQMD. Consulte los enlaces para obtener descripciones completas de los campos y sus valores disponibles.

Tabla 38. Nombres, descripciones y tipos de propiedades

Propiedad	Descripción	Tipo de Java	Enlace a la descripción completa
JMS_IBM_MQMD_Report	Opciones para mensajes de informe	Entero	Informe
JMS_IBM_MQMD_MsgType	Tipo de mensaje	Entero	MsgType
JMS_IBM_MQMD_Expiry	Duración del mensaje	Entero	Caducidad
JMS_IBM_MQMD_Feedback	Código de comentario o razón	Entero	FEEDBACK
JMS_IBM_MQMD_Encoding	Codificación numérica de datos de mensaje	Entero	Encoding
JMS_IBM_MQMD_CodedCharSetId	Identificador de juego de caracteres de datos de mensaje	Entero	CodedCharSetId
JMS_IBM_MQMD_Format	Nombre de formato de datos de mensaje	Serie	Formato
JMS_IBM_MQMD_Priority [1]	Prioridad de mensaje	Entero	Prioridad
JMS_IBM_MQMD_Persistence	Persistencia de los mensajes	Entero	Persistencia
JMS_IBM_MQMD_MsgId [2]	Identificador de mensaje	Object (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Identificador de correlación	Object (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	Contador de restitución	Entero	BackoutCount
JMS_IBM_MQMD_ReplyToQ	Nombre de la cola de respuestas	Serie	ReplyToQ

Tabla 38. Nombres, descripciones y tipos de propiedades (continuación)			
Propiedad	**Descripción**	**Tipo de Java**	**Enlace a la descripción completa**
JMS_IBM_MQMD_ReplyToQMgr	Nombre del gestor de colas de respuestas	Serie	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Identificador de usuario	Serie	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Señal de contabilidad	Object (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	Datos de aplicación relacionados con la identidad	Serie	ApplIdentityData
JMS_IBM_MQMD_PutApplType	Tipo de aplicación que coloca el mensaje	Entero	PutApplType
JMS_IBM_MQMD_PutApplName	Nombre de la aplicación que ha transferido el mensaje	Serie	PutApplName
JMS_IBM_MQMD_PutDate	Fecha cuando se colocó el mensaje	Serie	PutDate
JMS_IBM_MQMD_PutTime	Hora cuando se colocó el mensaje	Serie	PutTime
JMS_IBM_MQMD_ApplOriginData	Datos de aplicación relacionados con el origen	Serie	ApplOriginData
JMS_IBM_MQMD_GroupId	Identificador de grupo	Object (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	Número de secuencia del mensaje lógico en el grupo	Entero	MsgSeqNumber
JMS_IBM_MQMD_Offset	Desplazamiento de datos en el mensaje físico desde el inicio del mensaje lógico	Entero	desplazamiento
JMS_IBM_MQMD_MsgFlags	Distintivos de mensajes	Entero	MsgFlags
JMS_IBM_MQMD_OriginalLength	Longitud del mensaje original	Entero	OriginalLength

1. **Atención:** Si asigna un valor a JMS_IBM_MQMD_Priority que no está dentro del rango 0-9, esto vulnera la especificación JMS.
2. **Atención:** La especificación JMS establece que el ID de mensaje debe ser establecido por el proveedor de JMS y que debe ser exclusivo o nulo. Si asigna un valor a JMS_IBM_MQMD_MsgId, este valor se copia en JMSMessageID. Por lo tanto, no está establecido por el proveedor de JMS y puede que no sea exclusivo: esto vulnera la especificación JMS.
3. **Atención:** Si asigna un valor a JMS_IBM_MQMD_CorrelId que comienza con 'ID:', esto vulnera la especificación JMS.

4. **Atención:** El uso de propiedades de matriz de bytes en un mensaje vulnera la especificación JMS.

Acceso a los datos de un mensaje IBM MQ desde una aplicación que usa IBM MQ classes for JMS
Puede acceder a los datos completos de mensaje IBM MQ en una aplicación utilizando IBM MQ classes for JMS. Para acceder a todos los datos, el mensaje tiene que ser un `JMSBytesMessage`. El cuerpo del `JMSBytesMessage` incluye cualquier cabecera `MQRFH2`, todas las demás cabeceras IBM MQ y los siguientes datos de mensaje.

Establezca la propiedad `WMQ_MESSAGE_BODY` del destino a `WMQ_MESSAGE_BODY_MQ` para recibir todos los datos del cuerpo del mensaje en el `JMSBytesMessage`.

Si `WMQ_MESSAGE_BODY` está establecido en `WMQ_MESSAGE_BODY_JMS` o `WMQ_MESSAGE_BODY_UNSPECIFIED`, el cuerpo del mensaje se devuelve sin la cabecera JMS `MQRFH2` y las propiedades de `JMSBytesMessage` reflejan las propiedades establecidas en `RFH2`.

Algunas aplicaciones no pueden utilizar las funciones descritas en este tema. Si una aplicación está conectada con un gestor de colas IBM MQ V6, o si tiene establecida `PROVIDERVERSION` a 6, las funciones no estarán disponibles.

Envío de un mensaje

Cuando se envían mensajes, la propiedad de destino, `WMQ_MESSAGE_BODY`, tiene prioridad sobre `WMQ_TARGET_CLIENT`.

Si `WMQ_MESSAGE_BODY` está establecido en `WMQ_MESSAGE_BODY_JMS`, IBM MQ classes for JMS genera automáticamente una cabecera `MQRFH2` basándose en los valores de las propiedades `JMSMessage` y los campos de cabecera.

Si `WMQ_MESSAGE_BODY` se establece a `WMQ_MESSAGE_BODY_MQ`, no se añade ninguna cabecera adicional al cuerpo del mensaje.

Si `WMQ_MESSAGE_BODY` se establece a `WMQ_MESSAGE_BODY_UNSPECIFIED`, IBM MQ classes for JMS enviará una cabecera `MQRFH2` a menos que `WMQ_TARGET_CLIENT` esté establecida a `WMQ_TARGET_DEST_MQ`. En la recepción, si se establece `WMQ_TARGET_CLIENT` a `WMQ_TARGET_DEST_MQ`, da lugar a que se eliminen las `MQRFH2` del cuerpo del mensaje.

Nota: `JMSBytesMessage` y `JMSTextMessage` no requieren una `MQRFH2`, mientras que `JMSStreamMessage`, `JMSMapMessage` y `JMSObjectMessage`, sí.

`WMQ_MESSAGE_BODY_UNSPECIFIED` es el valor predeterminado de `WMQ_MESSAGE_BODY` y `WMQ_TARGET_DEST_JMS` es el valor predeterminado de `WMQ_TARGET_CLIENT`.

Si envía un `JMSBytesMessage`, puede alterar temporalmente los valores predeterminados para el cuerpo del mensaje JMS cuando se construye el mensaje IBM MQ. Utilice las propiedades siguientes:

- `JMS_IBM_Format` o `JMS_IBM_MQMD_Format`: esta propiedad especifica el formato de la cabecera IBM MQ o de la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.
- `JMS_IBM_Character_Set` o `JMS_IBM_MQMD_CodedCharSetId`: esta propiedad especifica el `CCSID` de la cabecera IBM MQ o la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.
- `JMS_IBM_Encoding` o `JMS_IBM_MQMD_Encoding`: esta propiedad especifica la codificación de la cabecera IBM MQ o la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.

Si se especifican ambos tipos de propiedad, las propiedades `JMS_IBM_MQMD_*` sustituirán las correspondientes propiedades `JMS_IBM_*`, siempre que la propiedad de destino `WMQ_MQMD_WRITE_ENABLED` esté establecida a `true`.

Las diferencias de efecto entre configurar las propiedades de mensaje con `JMS_IBM_MQMD_*` o `JMS_IBM_*` son significativas:

1. Las propiedades `JMS_IBM_MQMD_*` son específicas del proveedor IBM MQ JMS.

2. Las propiedades JMS_IBM_MQMD_* solo se configuran en la MQMD. Las propiedades JMS_IBM_* se establecen en MQMD solo si el mensaje no tiene una cabecera MQRFH2 JMS. De lo contrario, se establecen en la cabecera JMS RFH2.
3. Las propiedades JMS_IBM_MQMD_* no tienen ningún efecto sobre la codificación del texto y los números escritos en un JMSMessage.

   Es probable que una aplicación receptora que asume los valores de MQMD.Encoding y MQMD.CodedCharSetId se corresponda con la codificación y el juego de caracteres de los números y del texto del cuerpo del mensaje. Si se utilizan las propiedades JMS_IBM_MQMD_*, será responsabilidad de la aplicación emisora. La codificación y el conjunto de caracteres de números y texto en el cuerpo del mensaje se establecen mediante las propiedades JMS_IBM_*.

   El fragmento de código mal codificado en Figura 39 en la página 258 envía un mensaje codificado en el juego de caracteres 1208, con MQMD.CodedCharSetId establecido a 37.

   a. Envío de un mensaje incorrectamente codificado

```
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. Recepción del mensaje a partir del valor de JMS_IBM_CHARACTER_SET establecido por el valor de MQMD.CodedCharSetId:

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

   c. Salida resultante:

```
Message is "éÈÊ'>...??>?"
```

Figura 39. Datos de mensaje y MQMD codificados de forma incoherente

Cualquiera de los fragmentos de código de Figura 40 en la página 258 da como resultado un mensaje que se coloca en una cola o un tema, con un cuerpo que contiene la carga útil de la aplicación sin que se añada una cabecera MQRFH2 generada automáticamente.

1. Configuración de WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Configuración de WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                 setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Figura 40. Envío de un mensaje con un cuerpo de mensaje MQ.

Recepción de un mensaje

Si WMQ_MESSAGE_BODY se establece a WMQ_MESSAGE_BODY_JMS, el tipo y el cuerpo de mensaje entrante JMS se determinan a partir del contenido del mensaje WebSphere MQ recibido. El tipo de mensaje y el cuerpo están determinados por los campos de la cabecera MQRFH2, o en el MQMD si no hay ninguna MQRFH2.

Si WMQ_MESSAGE_BODY está establecido en WMQ_MESSAGE_BODY_MQ, el tipo de mensaje JMS de entrada es JMSBytesMessage. El cuerpo del mensaje JMS son los datos del mensaje devuelto por la llamada de API MQGET subyacente. La longitud del cuerpo del mensaje es la longitud devuelta por la llamada MQGET. El juego de caracteres y la codificación de los datos del cuerpo del mensaje están determinados por los campos CodedCharSetId y Encoding de MQMD. El formato de los datos del cuerpo del mensaje se determina mediante el campo Format de MQMD

Si WMQ_MESSAGE_BODY está establecida a WMQ_MESSAGE_BODY_UNSPECIFIED, el valor predeterminado, IBM MQ classes for JMS la establecerá a WMQ_MESSAGE_BODY_JMS.

Cuando se recibe un JMSBytesMessage, se puede decodificar consultando las propiedades siguientes:

- JMS_IBM_Format o JMS_IBM_MQMD_Format: esta propiedad especifica el formato de la cabecera IBM MQ o de la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.
- JMS_IBM_Character_Set o JMS_IBM_MQMD_CodedCharSetId: esta propiedad especifica el CCSID de la cabecera IBM MQ o la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.
- JMS_IBM_Encoding o JMS_IBM_MQMD_Encoding: esta propiedad especifica la codificación de la cabecera IBM MQ o la carga útil de la aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera WebSphere MQ anterior.

El siguiente fragmento de código resulta en la recepción de un mensaje JMSBytesMessage. Independientemente del contenido del mensaje recibido, y del campo de formato de la MQMD recibida, el mensaje será un JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Propiedad de destino WMQ_MESSAGE_BODY
WMQ_MESSAGE_BODY determina si una aplicación de JMS procesa la cabecera MQRFH2 de un mensaje de IBM MQ como parte de la carga útil del mensaje (es decir, como parte del cuerpo del mensaje de JMS).

Tabla 39. Nombres y descripciones de propiedades

Propiedad	Formato abreviado	Descripción
WMQ_MESSAGE_BODY	MBODY	Determina si una aplicación de JMS procesa la cabecera MQRFH2 de un mensaje de IBM MQ como parte de la carga útil del mensaje (es decir, como parte del cuerpo del mensaje de JMS).

Tabla 40. Nombres, valores y métodos set de propiedades			
Propiedad	**Valores válidos en herramienta de administración (valores predeterminados en negrita)**	**Valores válidos en programas**	**Método set**
WMQ_MESSAGE_CUERPO	• **UNSPECIFIED** En el envío, determina si IBM MQ classes for JMS genera e incluye una cabecera MQRFH2, dependiendo del valor de WMQ_TARGET_CLIENT. En la recepción, actúa como valor JMS. • JMS En el envío, IBM MQ classes for JMS genera automaticamente una cabecera MQRFH2 y la incluye en el mensaje de IBM MQ. En la recepción, IBM MQ classes for JMS establece las propiedades de mensajes de JMS de acuerdo con los valores contenidos en la cabecera MQRFH2 (si existe); no presenta la cabecera MQRFH2 como parte del cuerpo del mensaje JMS. • MQ En el envío, IBM MQ classes for JMS no genera una cabecera MQRFH2. En la recepción, IBM MQ classes for JMS presenta MQRFH2 como parte del cuerpo del mensaje de JMS.	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

Mensajes persistentes de JMS
Las aplicaciones de IBM MQ classes for JMS pueden utilizar el atributo de cola **NonPersistentMessageClass** para proporcionar un mejor rendimiento para los mensajes persistentes de JMS, a expensas de algo de fiabilidad.

Las colas de IBM MQ tienen un atributo denominado **NonPersistentMessageClass**. El valor de este atributo determina si se descartan los mensajes no persistentes de la cola cuando se reinicia el gestor de colas.

Puede establecer este atributo para una cola local utilizando el mandato IBM MQ Script (MQSC), DEFINE QLOCAL, con cualquiera de los dos parámetros siguientes:

NPMCLASS(NORMAL)
Los mensajes no persistentes de la cola se descartan cuando se reinicia el gestor de colas. Éste es el valor predeterminado.

NPMCLASS(HIGH)
Los mensajes no persistentes en la cola no se descartan cuando se reinicia el gestor de colas después de una conclusión progresiva o inmediata. Pero los mensajes no persistentes se pueden descartar después de un conclusión preferente o error.

Este tema describe cómo las aplicaciones de IBM MQ classes for JMS pueden utilizar este atributo de cola para proporcionar un mejor rendimiento para los mensajes persistentes de JMS.

La propiedad PERSISTENCE de un objeto Queue o Topic puede tener el valor HIGH. Puede utilizar la herramienta de administración de IBM MQ JMS para establecer este valor, o una aplicación puede llamar al método Destination.setPersistence() pasando el valor WMQConstants.WMQ_PER_NPHIGH como parámetro.

Si una aplicación envía un mensaje persistente de JMS o un mensaje no persistente de JMS a un destino cuya propiedad PERSISTENCE tiene el valor HIGH y la cola de IBM MQ subyacente se establece en NPMCLASS(HIGH), el mensaje se transfiere a la cola como mensaje no persistente de IBM MQ. Si la propiedad PERSISTENCE del destino no tiene el valor HIGH, o si la cola subyacente se establece en NPMCLASS(NORMAL), un mensaje persistente de JMS se coloca en la cola como un mensaje persistente de IBM MQ, y un mensaje no persistente de JMS se coloca en la cola como mensaje no persistente de IBM MQ.

Si un mensaje persistente de JMS se coloca en una cola como un mensaje no persistente de IBM MQ y desea asegurarse de que el mensaje no se descarte después de una conclusión progresiva o inmediata de un gestor de colas, todas las colas a través de las que se pueda encaminar el mensaje se deben establecer en NPMCLASS(HIGH). En el dominio de publicación/suscripción, estas colas incluyen colas de suscriptor. Como ayuda para aplicar esta configuración, IBM MQ classes for JMS emite una excepción InvalidDestinationException si una aplicación intenta crear un consumidor de mensajes para un destino donde la propiedad PERSISTENCE tiene el valor HIGH y la cola subyacente de IBM MQ está establecida en NPMCLASS(NORMAL).

El establecimiento de la propiedad PERSISTENCE de un destino en HIGH no afecta a la forma en que se recibe un mensaje desde ese destino. Un mensaje enviado como un mensaje persistente de JMS se recibe como mensaje persistente de JMS, y un mensaje enviado como un mensaje no persistente de JMS se recibe como mensaje no persistente de JMS .

Cuando una aplicación envía el primer mensaje a un destino donde la propiedad PERSISTENCE tiene el valor HIGH, o cuando una aplicación crea el primer consumidor de mensajes para un destino donde la propiedad PERSISTENCE tiene el valor HIGH, IBM MQ classes for JMS emite una llamada MQINQ para determinar si NPMCLASS(HIGH) está establecido en la cola subyacente de IBM MQ. Por consiguiente, la aplicación debe tener la autorización para consultar la cola. Además, IBM MQ classes for JMS conserva el resultado de la llamada MQINQ hasta que se suprime el destino, y no emite más llamadas MQINQ. Por lo tanto, si cambia el valor de NPMCLASS en la cola subyacente mientras la aplicación todavía está utilizando el destino, IBM MQ classes for JMS no reconoce el nuevo valor.

Al permitir que los mensajes persistentes de JMS se coloquen en colas IBM MQ como mensajes IBM MQ no persistentes, se gana en rendimiento a expensas de algo de fiabilidad. Si necesita la máxima fiabilidad para los mensajes persistentes de JMS, no envíe los mensajes a un destino donde la propiedad PERSISTENCE tenga el valor HIGH.

La capa de JMS puede utilizar SYSTEM.JMS.TEMPQ.MODEL, en lugar de SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEM.JMS.TEMPQ.MODEL crea colas dinámicas permanentes que aceptan mensajes persistentes, porque SYSTEM.DEFAULT.MODEL.QUEUE no puede aceptar mensajes persistentes. Para utilizar colas temporales para aceptar mensajes persistentes, debe por lo tanto utilizar SYSTEM.JMS.TEMPQ.MODEL, o cambiar la cola de modelo a una cola alternativa de su elección.

Utilización de TLS con IBM MQ classes for JMS
Las aplicaciones de IBM MQ classes for JMS pueden utilizar el cifrado de TLS (Transport Layer Security). Para ello necesitan un proveedor JSSE.

Las conexiones de IBM MQ classes for JMS en las que se utiliza TRANSPORT(CLIENT) permiten el cifrado de TLS. TLS proporciona cifrado de la comunicación, autenticación e integridad de los mensajes. Se suele

utilizar para proteger las comunicaciones entre dos interlocutores cualesquiera en Internet o dentro de una intranet.

IBM MQ classes for JMS utiliza Java Secure Socket Extension (JSSE) para gestionar el cifrado de TLS y por tanto necesita un proveedor JSSE. Las JVM de JSE v1.4 tienen un proveedor JSSE incorporado. Los detalles acerca de la gestión y almacenamiento de certificados pueden variar en función del proveedor. Si desea obtener información sobre este tema, consulte la documentación del proveedor de JSSE.

En este apartado se da por supuesto que el proveedor JSSE está instalado y configurado correctamente, y que se han instalado los certificados pertinentes y se han puesto a la disposición del proveedor JSSE. Ahora puede utilizar JMSAdmin para establecer varias propiedades administrativas.

Si la aplicación de IBM MQ classes for JMS utiliza una tabla de definición de canal de cliente (CCDT) para conectar con un gestor de colas, consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS" en la página 292.

Propiedad de objeto SSLCIPHERSUITE
Establezca SSLCIPHERSUITE para habilitar el cifrado TLS en un objeto ConnectionFactory.

Para habilitar el cifrado TLS en un objeto ConnectionFactory, utilice JMSAdmin para establecer la propiedad SSLCIPHERSUITE en una CipherSuite soportada por el proveedor JSSE. Debe coincidir con la CipherSpec establecida en el canal de destino. Sin embargo, CipherSuites son distintas de las CipherSpecs y por consiguiente, tienen nombres distintos. La sección "CipherSpecs y CipherSuites de TLS en IBM MQ classes for JMS" en la página 265 contiene una tabla que correlaciona las CipherSpecs soportadas por IBM MQ con las CipherSuites equivalentes tal como las conoce JSSE. Para obtener más información sobre CipherSpecs y CipherSuites con IBM MQ, consulte Protección IBM MQ.

Por ejemplo, para configurar un objeto ConnectionFactory que se puede utilizar para crear una conexión a través de un canal MQI habilitado para TLS con una CipherSpec denominada TLS_RSA_WITH_AES_128_CBC_SHA, emita el mandato siguiente para JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Esto se puede establecer también desde una aplicación, mediante el método setSSLCipherSuite() en un objeto MQConnectionFactory.

Para mayor comodidad, si se especifica una CipherSpec en la propiedad SSLCIPHERSUITE, JMSAdmin intenta correlacionar la CipherSpec con la CipherSuite correspondiente y emite un aviso. Este intento de correlación no se realiza si la propiedad la especifica una aplicación.

Como alternativa, utilice la tabla de definición de canal de cliente (CCDT). Para obtener más información, consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS" en la página 292.

propiedad de objeto SSLFIPSREQUIRED
Si necesita una conexión para utilizar una CipherSuite que esté soportada por el proveedor de IBM Java JSSE FIPS (IBMJSSEFIPS), establezca la propiedad SSLFIPSREQUIRED de la fábrica de conexiones en YES.

Nota: En AIX, Linux, and Windows, IBM MQ proporciona conformidad con FIPS 140-2 a través del módulo criptográfico IBM Crypto for C (ICC) . El certificado para este módulo se ha movido al estado Histórico. Los clientes deben ver el certificado deIBM Crypto for C (ICC) y tener en cuenta cualquier consejo proporcionado por NIST. Un módulo FIPS 140-3 de sustitución está actualmente en curso y su estado se puede ver buscándolo en los módulos NIST CMVP en la lista de procesos.

El valor predeterminado de esta propiedad es NO, lo que significa que una conexión puede utilizar cualquier CipherSuite que esté soportada por IBM MQ.

Si una aplicación utiliza más de una conexión, el valor del campo sslFipsRequired que se utiliza cuando la aplicación crea la primera conexión determina el valor que se utiliza cuando la aplicación crea cualquier conexión posterior. Esto significa que se hace caso omiso del valor de la propiedad SSLFIPSREQUIRED de la fábrica de conexiones que se utiliza para crear una conexión posterior. Debe reiniciar la aplicación si desea utilizar un valor diferente para el campo SSLFIPSREQUIRED.

Una aplicación puede establecer esta propiedad invocando el método setSSLFipsRequired() de un objeto ConnectionFactory. Se hace caso omiso de esta propiedad si no se ha establecido ninguna CipherSuite.

Tareas relacionadas

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Referencia relacionada

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

propiedad de objeto SSLPEERNAME

Utilice SSLPEERNAME para especificar un patrón de nombre distinguido a fin de asegurarse de que la aplicación JMS se conecta al gestor de colas correcto.

Una aplicación JMS se puede asegurar de que se conecta al gestor de colas correcto especificando un patrón de nombre distinguido. La conexión se establece correctamente si el gestor de colas presenta un nombre distinguido que coincide con el patrón. Para conocer detalles sobre el formato de este patrón, consulte los temas relacionados.

El nombre distinguido se establece utilizando la propiedad SSLPEERNAME de un objeto ConnectionFactory. Por ejemplo, el mandato JMSAdmin siguiente configura un objeto ConnectionFactory de forma que el gestor de colas se deba identificar con un nombre común que empiece con los caracteres `QMGR.` y contenga al menos dos nombres de unidades organizativas, el primero de los cuales debe ser IBM y el segundo WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

La comprobación no es sensible a las mayúsculas y minúsculas y se puede utilizar el carácter de punto y coma en lugar de comas. SSLPEERNAME también puede establecerse desde una aplicación utilizando el método setSSLPeerName() en un objeto MQConnectionFactory. Si no se establece esta propiedad, no se realiza ningún tipo de comprobación del nombre distinguido que suministra el gestor de colas. Si no se establece CipherSuiteSe, se hace caso omiso de esta propiedad.

Propiedad de objeto SSLCERTSTORES

Utilice SSLCERTSTORES para especificar una lista de servidores LDAP para utilizar la comprobación de lista de revocación de certificado (CRL).

Es habitual utilizar una lista de revocación de certificados (CRL) para identificar certificados que ya no son de confianza. Las CRL normalmente se alojan en servidores LDAP. JMS permite especificar un servidor LDAP para la comprobación de la CRL cuando se utiliza Java 2 v1.4 o posterior. El ejemplo siguiente de JMSAdmin hace que JMS utilice una CRL alojada en un servidor LDAP denominado crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Nota: Para utilizar un CertStore correctamente con una CRL alojada en un servidor LDAP, asegúrese de que el SDK (Software Development Kit) de Java es compatible con la CRL. Algunos SDK necesitan que la CRL cumpla el RFC 2587, el cual define un esquema para LDAP v2. La mayoría de servidores LDAP v3 utilizan el RFC 2256.

Si el servidor LDAP no se ejecuta en el puerto predeterminado 389, puede especificar el puerto añadiendo un símbolo de dos puntos (:) y el número de puerto al nombre de host. Si el certificado presentado por el gestor de colas está presente en el CRL alojado en crl1.ibm.com, no se realizará la conexión. Para evitar un punto único de anomalía, JMS permite proporcionar varios servidores LDAP especificando una lista de servidores LDAP separados por un espacio. He aquí un ejemplo:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Cuando se especifican varios servidores LDAP, JMS prueba secuencialmente cada uno de ellos hasta que encuentra un servidor con el que puede verificar satisfactoriamente el certificado del gestor de colas. Cada servidor debe contener la misma información.

Una aplicación del método MQConnectionFactory.setSSLCertStores() puede suministrar una serie de este formato. De forma alternativa, la aplicación puede crear uno o más objetos java.security.cert.CertStore, situarlos en un objeto Collection adecuado y suministrar este objeto Collection al método setSSLCertStores(). De esta forma, la aplicación puede personalizar la comprobación CRL. Consulte la documentación JSSE para obtener detalles sobre cómo crear y utilizar objetos CertStore.

El certificado que presenta el gestor de colas en el momento de establecer una conexión, se valida de la forma siguiente:

1. El primer objeto CertStore de la recopilación identificado como sslCertStores se utiliza para identificar un servidor CRL.
2. Se hace un intento de contactar con el servidor CRL.
3. Si resulta satisfactorio, se busca una coincidencia del certificado en el servidor.
   a. Si se encuentra el certificado para revocarlo, finaliza el proceso de búsqueda y la petición de conexión no responde indicando el código de razón MQRC_SSL_CERTIFICATE_REVOKED.
   b. Si no se encuentra el certificado, finaliza el proceso de búsqueda y se permite que la conexión continúe.
4. Si el intento de contactar con el servidor no resulta satisfactorio, se utiliza el siguiente objeto CertStore para identificar un servidor CRL y se repite el proceso desde el paso 2.

   Si se trataba del último CertStore de la recopilación, o si la recopilación no contiene objetos CertStore, el proceso de búsqueda no responde y la petición de conexión no responde, indicando el código de razón MQRC_SSL_CERT_STORE_ERROR.

El objeto Collection determina el orden en el que se utilizan los CertStores.

Si la aplicación utiliza setSSLCertStores() para establecer una recopilación de objetos CertStore, MQConnectionFactory ya no podrá enlazarse más a un espacio de nombres JNDI. Si intenta hacerlo, se generará una excepción. Si no establece sslCertStores correctamente, no se realiza ningún tipo de comprobación de revocaciones del certificado que suministra el gestor de colas. Si no se establece CipherSuiteSe, se hace caso omiso de esta propiedad.

Propiedad de objeto SSLRESETCOUNT
Esta propiedad representa el número total de bytes que envía y recibe una conexión antes de que se renegocie la clave secreta utilizada para cifrado.

El número de bytes enviados es el número antes del cifrado y el número de bytes recibidos es el número después del cifrado. El número de bytes también incluye información de control enviada y recibida por IBM MQ classes for JMS.

Por ejemplo, para configurar un objeto ConnectionFactory que se puede utilizar para crear una conexión sobre un canal MQI habilitado para TLS, con una clave secreta que se renegocia después de que se hayan enviado 4 MB de datos, emita el siguiente mandato a JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Una aplicación puede establecer esta propiedad invocando el método setSSLResetCount() de un objeto ConnectionFactory.

Si el valor de esta propiedad es cero, que es el valor predeterminado, la clave secreta nunca se renegocia. Se hace caso omiso de esta propiedad si no se ha establecido ninguna CipherSuite.

La propiedad de objeto SSLSocketFactory
Para personalizar otros aspectos de la conexión TLS para una aplicación, cree un SSLSocketFactory y configure JMS para utilizarlo.

Puede personalizar otros aspectos de la conexión TLS para una aplicación. Por ejemplo, puede desear inicializar hardware criptográfico o cambiar el almacén de confianza y el almacén de claves que están en uso. Para ello, la aplicación debe crear primero un objeto javax.net.ssl.SSLSocketFactory que esté personalizado debidamente. Consulte la documentación de JSSE para obtener información sobre

cómo hacer esto, pues las funciones personalizables varían según el proveedor. Una vez obtenido un objeto SSLSocketFactory adecuado, utilice el método MQConnectionFactory.setSSLSocketFactory() para configurar JMS para que utilice el objeto SSLSocketFactory personalizado.

Si la aplicación utiliza el método setSSLSocketFactory() para establecer un objeto SSLSocketFactory personalizado, el objeto MQConnectionFactory ya no podrá enlazarse a un espacio de nombres JNDI. Si intenta hacerlo, se generará una excepción. Si esta propiedad no está establecida, se utiliza el objeto SSLSocketFactory predeterminado. Consulte la documentación de JSSE para obtener detalles del comportamiento del objeto predeterminado SSLSocketFactory. Si no se establece CipherSuiteSe, se hace caso omiso de esta propiedad.

Importante: Observe que el uso de las propiedades de SSL no garantiza la seguridad cuando un objeto ConnectionFactory se recupera de un espacio de nombres JNDI que en sí mismo no es seguro. En concreto, la implementación estándar de JNDI por LDAP no es segura. Un atacante puede suplantar al servidor LDAP y hacer que una aplicación de JMS se conecte a un servidor incorrecto sin que la aplicación tenga conocimiento de ello. Si están establecidas las medidas de seguridad adecuadas, otras implementaciones de JNDI (tal como la implementación fscontext) son seguras.

Realización de cambios en el almacén de claves o almacén de confianza de JSSE
Si realiza cambios en el almacén de claves o almacén de confianza, debe emprender determinadas acciones para que los cambios entren en vigor.

Si cambia el contenido del almacén de claves o del almacén de confianza de JSSE, o cambia la ubicación del archivo de almacén de claves o de almacén de confianza, las aplicaciones de IBM MQ classes for JMS que se estén ejecutando en ese momento no recogerán automáticamente los cambios. Para que los cambios surtan efecto, deben realizarse las acciones siguientes:

- Las aplicaciones deben cerrar todas sus conexiones y eliminar cualquier conexión sin utilizar en las agrupaciones de conexiones.
- Si el proveedor JSSE almacena información del almacén de claves y almacén de confianza, esta información se debe actualizar.

Una vez realizadas estas acciones, las aplicaciones pueden volver a crear sus conexiones.

Dependiendo de cómo estén diseñadas las aplicaciones y de la función proporcionada por el proveedor JSSE, puede ser posible realizar estas acciones sin detener y reiniciar las aplicaciones. Pero detener y reiniciar las aplicaciones puede ser la solución más sencilla.

CipherSpecs y CipherSuites de TLS en IBM MQ classes for JMS
La capacidad de las aplicaciones de IBM MQ classes for JMS para establecer conexiones con un gestor de colas depende de la suite de cifrado especificada en el extremo servidor del canal MQI y de la suite de cifrado especificada en el extremo cliente.

La tabla siguiente lista las especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes.

Deprecated Revise el tema CipherSpecs en desuso para ver si alguna de las especificaciones de cifrado listadas en la tabla siguiente ha dejado de ser utilizada por IBM MQ y, si es así, en qué actualización se ha dejado de utilizar la especificación de cifrado.

Importante: Las CipherSuites listadas son las soportadas por el IBM Java Runtime Environment (JRE) proporcionado con IBM MQ. Las suites de cifrado indicadas en la lista incluyen las soportadas por el Java JRE de Oracle. Para obtener más información sobre cómo configurar la aplicación para utilizar un Oracle Java JRE, consulte Configuración de la aplicación para utilizar correlaciones de CipherSuite IBM Java o Oracle Java.

La tabla también incluye el protocolo utilizado por la aplicación y la indicación de si la suite de cifrado cumple el estándar FIPS 140-2.

Nota: En AIX, Linux, and Windows, IBM MQ proporciona conformidad con FIPS 140-2 a través del módulo criptográfico IBM Crypto for C (ICC) . El certificado para este módulo se ha movido al estado Histórico. Los clientes deben ver el certificado deIBM Crypto for C (ICC) y tener en cuenta cualquier consejo

proporcionado por NIST. Un módulo FIPS 140-3 de sustitución está actualmente en curso y su estado se puede ver buscándolo en los módulos NIST CMVP en la lista de procesos.

Las suites de cifrado designadas como compatibles con FIPS 140-2 se pueden utilizar si la aplicación no se ha configurado para aplicar la conformidad con FIPS 140-2, pero si se ha configurado la conformidad con FIPS 140-2 para la aplicación (consulte las notas siguientes sobre la configuración) sólo se pueden configurar las suites de cifrado marcadas como compatibles con FIPS 140-2; el intento de utilizar otras suites de cifrado produce un error.

Nota: Cada JRE puede tener varios proveedores de seguridad criptográfica, cada uno de los cuales puede contribuir a una implementación de la misma CipherSuite. Pero no todos los proveedores de seguridad están certificados como compatibles con FIPS 140-2. Si la conformidad con FIPS 140-2 no se aplica para una aplicación, es posible que se utilice una implementación no certificada de la suite de cifrado. Las implementaciones no certificadas pueden no trabajar en conformidad con FIPS 140-2, incluso si la suite de cifrado cumple teóricamente el nivel de seguridad mínimo exigido por el estándar. Consulte las notas siguientes para obtener más información acerca de cómo la configurar la imposición de FIPS 140-2 en las aplicaciones IBM MQ JMS.

Para obtener más información acerca de la conformidad con FIPS 140-2 y Suite-B para CipherSpecs y CipherSuites, consulte Especificación de CipherSpecs. También puede ser necesario que conozca información que se refiere a los Estándares federales de proceso de información de los Estados Unidos.

Para utilizar el conjunto completo de suites de cifrado y trabajar en conformidad con FIPS 140-2 o Suite-B, es necesario un JRE adecuado. adecuado.IBM Java 7 Service Refresh 4 Fixpack 2 o un nivel superior de IBM JRE proporciona el soporte adecuado para las CipherSuites TLS 1.2 listadas en Tabla 41 en la página 267.

Para poder utilizar cifrados TLS 1.3 , el JRE que ejecuta la aplicación debe dar soporte a TLS 1.3.

Nota: Para utilizar algunas suites de cifrado, es necesario configurar los archivos de política 'no restringidos' en el JRE. Para obtener más detalles sobre cómo se configuran los archivos de políticas en un SDK o JRE, consulte el tema *Archivos de políticas SDK de IBM* en la publicación *Security Reference for IBM SDK, Java Technology Edition* correspondiente a la versión que está utilizando.

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	No
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sí

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sí
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" en la página 286	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	no "4" en la página 286

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	no "4" en la página 286
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	no "4" en la página 286

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	no "4" en la página 286
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	no "4" en la página 286

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	no "4" en la página 286
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	no "4" en la página 286

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	No
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	No
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sí
TLS_AES_128_GCM_SHA256 "3" en la página 286	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_AES_256_GCM_SHA384 "3" en la página 286	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	No
TLS_CHACHA20_POLY1305_SHA256 "3" en la página 286	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_AES_128_CCM_SHA256 "3" en la página 286	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	No
TLS_AES_128_CCM_8_SHA256 "3" en la página 286	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	No
Cualquiera "3" en la página 286	*ANY	*ANY	Múltiple	No
ANY_TLS13 "3" en la página 286	*TLS13	*TLS13	TLS V13	No

Tabla 41. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 286**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ANY_TLS12_OR_HIGHER "3" en la página 286	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 y superior	No
ANY_TLS13_OR_HIGHER "3" en la página 286	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 y superior	No

Notas:

1. Este es el valor configurado en un canal en IBM MQ, incluido en una CCDT (binario o JSON).
2. Deprecated La CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.
3. Para poder utilizar cifrados TLS v1.3 , el Java runtime environment (JRE) que ejecuta la aplicación debe dar soporte a TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 A partir de IBM MQ 9.3.5 CSU 1 y IBM MQ 9.3.0 CSU 17, el JRE de IBM Java 8 elimina el soporte para el intercambio de claves RSA cuando opera en modalidad FIPS.

Configuración de suites de cifrado y la conformidad con FIPS en una aplicación de IBM MQ classes for JMS

- Una aplicación que utiliza IBM MQ classes for JMS puede utilizar cualquiera de dos métodos para establecer la suite de cifrado para una conexión:
  - Llamar al método setSSLCipherSuite de un objeto ConnectionFactory.
  - Utilizar la herramienta de administración IBM MQ JMS para establecer la propiedad SSLCIPHERSUITE de un objeto ConnectionFactory.
- Una aplicación que utiliza IBM MQ classes for JMS puede utilizar cualquiera de dos métodos para aplicar la conformidad con FIPS 140-2:
  - Llamar al método setSSLFipsRequired de un objeto ConnectionFactory.
  - Utilizar la herramienta de administración IBM MQ JMS para establecer la propiedad SSLFIPSREQUIRED de un objeto ConnectionFactory.

Configuración de la aplicación para utilizar correlaciones de las suites de cifrado de IBM Java u Oracle Java

Nota: V 9.3.3 Para Continuous Delivery de IBM MQ 9.3.3, la Java Propiedad del sistema `com.ibm.mq.cfg.useIBMCipherMappings`, que controla qué correlaciones se utilizan, se elimina del producto. A partir de IBM MQ 9.3.3, un cifrado se puede definir como el nombre CipherSpec o CipherSuite y IBM MQlo maneja correctamente. Los tres archivos JAR Jackson siguientes son necesarios si la aplicación IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging crea conexiones TLS seguras con un gestor de colas:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

Importante: La información siguiente sobre `com.ibm.mq.cfg.useIBMCipherMappings` se aplica únicamente a Long Term Support y Continuous Delivery antes de IBM MQ 9.3.3 .

Puede configurar si la aplicación utiliza las correlaciones predeterminadas de suites de cifrado de IBM Java con especificaciones de cifrado de IBM MQ, o las correlaciones de suites de cifrado de Oracle con especificaciones de cifrado de IBM MQ. Por lo tanto, puede utilizar suites de cifrado de TLS si la aplicación utiliza un JRE de IBM o un JRE de Oracle. La Java Propiedad del sistema `com.ibm.mq.cfg.useIBMCipherMappings` controla qué correlaciones se utilizan. La propiedad puede tener uno de los valores siguientes:

true
Utilizar las correlaciones de suites de cifrado de IBM Java con especificaciones de cifrado de IBM MQ.

Este es el valor predeterminado.

falso
Utilizar las correlaciones de suites de cifrado de Oracle con especificaciones de cifrado de IBM MQ.

Para obtener más información sobre cómo utilizar IBM MQ Java y los cifrados TLS, consulte la publicación de blog de MQdev MQ Java, Cifrados TLS, NoIBM JRE & APARs IT06775, IV66840, IT09423, IT10837.

Limitaciones de interoperatividad

Determinadas suites de cifrado pueden ser compatibles con más de una especificación de cifrado de IBM MQ, dependiendo del protocolo que se esté utilizando. Pero solo está soportada la combinación CipherSuite/CipherSpec que utiliza la versión de TLS especificada en la Tabla 1. El intento de utilizar combinaciones no soportadas de suites de cifrado y especificaciones de cifrado producirá un error y se devolverá la excepción correspondiente. Las instalaciones que utilicen cualquiera de estas combinaciones de suite de cifrado/especificación de cifrado se deben trasladar a una combinación soportada.

La tabla siguiente muestra las suites de cifrado a la que se aplica esta limitación.

Tabla 42. Suites de cifrado y sus correspondientes especificaciones de cifrado soportadas y no soportadas		
CipherSuite	**Especificación de cifrado soportada para TLS**	**Especificación de cifrado no soportada para TLS**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" en la página 288	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated La especificación de cifrado TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.

Escritura de salida de canal en Java para IBM MQ classes for JMS
Puede crear salidas de canal definiendo clases Java que implementen las interfaces especificadas.

Para obtener una introducción a las salidas de seguridad, empiece con el tema Programas de salida de seguridad de canal .

En el paquete com.ibm.mq.exits hay tres interfaces definidas:

- WMQSendExit, para una salida de emisión
- WMQReceiveExit, para una salida de recepción
- WMQSecurityExit, para una salida de seguridad

El código de ejemplo siguiente define una clase que implementa las tres interfaces:

```
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
  }
}
```

A cada salida recibe un objeto MQCXP y un objeto MQCD como parámetros. Estos objetos representan las estructuras MQCXP y MQCD definidas en la interfaz de procedimientos.

Cuando se invoca una salida de emisión, el parámetro agentBuffer contiene los datos que está a punto de enviar al gestor de colas del servidor. No es necesario un parámetro de longitud, pues la expresión agentBuffer.limit() proporciona la longitud de los datos. La salida de emisión devuelve como su valor los datos que se deben enviar al gestor de colas del servidor. No obstante, si la salida de emisión no es la última salida de emisión de una secuencia de salidas de emisión, en su lugar, los datos devueltos se pasan a la siguiente salida de emisión de la secuencia. Una salida de emisión puede devolver una versión modificada de los datos que recibe en el parámetro agentBuffer o puede devolver datos sin modificar. Por lo tanto, el cuerpo de salida más simple posible es:

```
{ return agentBuffer; }
```

Cuando se invoca una salida de recepción, el parámetro agentBuffer contiene los datos que se han recibido del gestor de colas del servidor. La salida de recepción devuelve como su valor los datos que IBM MQ classes for JMS debe pasar a la aplicación. No obstante, si la salida de recepción no es la última salida de recepción de una secuencia de salidas de recepción, en su lugar, los datos devueltos se pasan a la siguiente salida de recepción de la secuencia.

Cuando se invoca una salida de seguridad, el parámetro agentBuffer contiene los datos que se han recibido en un flujo de seguridad de la salida de seguridad en el extremo del servidor de la conexión. La salida de seguridad devuelve como su valor los datos que se han de enviar en un flujo de seguridad a la salida de seguridad del servidor.

Las salidas de canal se invocan con un almacenamiento intermedio que tiene una matriz de seguridad. Para obtener el mejor rendimiento, la salida debe devolver un almacenamiento intermedio con una matriz auxiliar.

Se pueden pasar hasta un máximo de 32 caracteres a una salida de canal cuando se invoca. La salida accede a los datos de usuario llamando al método getExitData() del objeto MQCXP. Aunque la salida puede cambiar los datos de usuario llamando al método setExitData(), los datos de usuario se renuevan cada vez que se invoca la salida. Por consiguiente, los cambios efectuados en los datos de usuario se perderán. No obstante, la salida puede pasar datos de una llamada a la siguiente utilizando el área de usuario de la salida del objeto MQCXP. La salida accede al área de usuario de la salida por referencia llamando al método getExitUserArea().

Cada clase de salida debe tener un constructor. El constructor puede ser el constructor predeterminado, como se muestra en el ejemplo anterior, o puede ser un constructor con un parámetro de serie. El constructor se invoca para crear una instancia de la clase de salida para cada salida definida en la clase. Por lo tanto, en el ejemplo anterior, se crea una instancia de la clase MyMQExits para la salida de emisión, se crea otra instancia para la salida de recepción y se crea una tercera instancia para salida de seguridad. Cuando se invoca un constructor con un parámetro de serie, el parámetro contiene los mismos datos de usuario que se han pasado a la salida de canal para la que se está creando la instancia. Si una clase de salida tiene un constructor predeterminado y también un constructor de un solo parámetro, el constructor de un solo parámetro tiene prioridad.

No cierre la conexión desde dentro de una salida de canal.

Cuando se envían datos al extremo servidor de una conexión, el cifrado TLS se realiza *después* de invocar cualquier salida de canal. Del mismo modo, cuando se reciben datos desde el extremo servidor de una conexión, el descifrado TLS se realiza *antes* de invocar cualquier salida de canal.

En las versiones de IBM MQ classes for JMS anteriores a IBM WebSphere MQ 7.0, las salidas de canal se implementan utilizando las interfaces MQSendExit, MQReceiveExit y MQSecurityExit. Puede seguir utilizando estas interfaces, pero se prefieren las nuevas interfaces para obtener un rendimiento y funcionamiento mejores.

Configuración de IBM MQ classes for JMS para utilizar salidas de canal
Una aplicación de IBM MQ classes for JMS puede utilizar salidas de seguridad de canal, salidas de emisión y salidas de recepción en el canal MQI que se inicia cuando la aplicación se conecta a un gestor de colas. La aplicación puede utilizar salidas escritas en Java, C o C++. La aplicación también puede utilizar una secuencia de salidas de envío o recepción que se ejecutan sucesivamente.

Las propiedades siguientes se utilizan para especificar una salida de emisión o una secuencia de salidas de emisión que son utilizadas por una conexión JMS:

- La propiedad **SENDEXIT** de un objeto MQConnectionFactory.
- La propiedad **sendexit** contenida en una especificación de activación utilizada por el adaptador de recursos de IBM MQ para la comunicación entrante.
- La propiedad **sendexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM MQ para la comunicación de salida.

El valor de la propiedad es una serie de caracteres que comprende uno o varios elementos separados por comas. Cada elemento identifica una salida de emisión de una de las maneras siguientes:

- El nombre de una clase que implementa la interfaz `WMQSendExit` para una salida de emisión escrita en Java.
- Una serie de caracteres con el formato *nombreBiblioteca* (*nombrePuntoEntrada*) para una salida de emisión escrita en C o C++.

Del mismo modo, las propiedades siguientes especifican la salida de recepción o secuencia de salidas de recepción utilizadas por una conexión.

- La propiedad **RECEXIT** de un objeto MQConnectionFactory.
- La propiedad **receiveexit** contenida en una especificación de activación utilizada por el adaptador de recursos de IBM MQ para la comunicación entrante.
- La propiedad **receiveexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM MQ para la comunicación de salida.

Las propiedades siguientes especifican la salida de seguridad utilizada por una conexión:

- La propiedad **SECEXIT** de un objeto MQConnectionFactory.
- La propiedad **securityexit** contenida en una especificación de activación utilizada por el adaptador de recursos de IBM MQ para la comunicación entrante.
- La propiedad **securityexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM MQ para la comunicación de salida.

Para MQConnectionFactories, puede establecer las propiedades **SENDEXIT**, **RECEXIT** y **SECEXIT** utilizando la herramienta de administración de IBM MQ JMS o IBM MQ Explorer. Como alternativa, una aplicación puede establecer esas propiedades invocando los métodos `setSendExit()`, `setReceiveExit()` y `setSecurityExit()`.

Las salidas de canal se cargan mediante su propio cargador de clases. Para encontrar una salida de canal, el cargador de clases busca en las ubicaciones siguientes en el orden especificado:

1. La vía de acceso de clases especificada por la propiedad **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** o por el atributo **JavaExitsClassPath** en la stanza Channels del archivo de configuración de cliente de IBM MQ.
2. Deprecated La vía de acceso de clase especificada por la propiedad del sistema Java **com.ibm.mq.exitClasspath**. Observe que esta propiedad ahora ya no se utiliza.
3. El directorio exits de IBM MQ, tal como se muestra en la Tabla 43 en la página 290. El cargador de clases primero busca en el directorio los archivos de clase que no están empaquetados en los archivos JAR de Java. Si no encuentra la salida de canal, el cargador de clases busca los archivos JAR en el directorio.

Tabla 43. El directorio exits de IBM MQ

Plataforma	Directorio
Linux AIX AIX and Linux	`/var/mqm/exits` (salidas de canal de 32 bits) `/var/mqm/exits64` (salidas de canal de 64 bits)

Tabla 43. El directorio exits de IBM MQ (continuación)	
Plataforma	**Directorio**
Windows	*dir_datos_instalación*\exits donde *dir_datos_instalación* es el directorio que eligió para los archivos de datos de IBM MQ durante la instalación. El directorio predeterminado es C:\ProgramData\IBM\MQ.

Nota: Si existe una salida de canal en más de una ubicación, IBM MQ classes for JMS carga la primera instancia que encuentra.

El padre del cargador de clases es el cargador de clases que se utiliza para cargar IBM MQ classes for JMS. Por lo tanto, es posible que el cargador de clases padre cargue una salida de canal si no se puede encontrar en ninguna de las ubicaciones anteriores. Sin embargo, cuando utiliza IBM MQ classes for JMS en un entorno como, por ejemplo, un servidor de aplicaciones JEE, no es probable que pueda influir en la elección del cargador de clases padre, por lo que el cargador de clases se debe configurar estableciendo la propiedad de sistema Java **`com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath`** en el servidor de aplicaciones.

Si la aplicación se está ejecutando con Java security manager habilitado, el archivo de configuración de política utilizado por el entorno de ejecución Java donde se ejecuta la aplicación debe tener los permisos para cargar una clase de salida de canal. Para obtener información sobre cómo hacerlo, consulte Ejecución de clases de IBM MQ para aplicaciones JMS en el Java Gestor de seguridad.

Las interfaces `MQSendExit`, `MQReceiveExity` `MQSecurityExit` suministradas con versiones anteriores a IBM WebSphere MQ 7.0 siguen estando soportadas. Si utiliza salidas de canal que implementan estas interfaces, `com.ibm.mq.jar` debe estar presente en la vía de acceso de clases.

Para obtener información sobre cómo escribir salidas de canal en C, consulte “Programas de salida de canal para canales de mensajes” en la página 986. Los programas de salida de canal escritos en C o C++ se deben almacenar en el directorio mostrado en la Tabla 43 en la página 290.

Si la aplicación utiliza una tabla de definición de canal de cliente (CCDT) para conectar con un gestor de colas, consulte “Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS” en la página 292.

Especificación de los datos de usuario que se deben pasar a salidas de canal cuando se utiliza IBM MQ classes for JMS
Se pueden pasar hasta un máximo de 32 caracteres a una salida de canal cuando se invoca.

La propiedad SENDEXITINIT de un objeto MQConnectionFactory especifica los datos de usuario que se pasan a cada salida de emisión cuando esta se invoca. El valor de la propiedad es una serie de caracteres que comprende uno o varios elementos de datos de usuario separados por comas. La posición de cada elemento de datos de usuario dentro de la serie de caracteres determina a qué salida de emisión, en una secuencia de salidas de emisión, se pasan los datos de usuario. Por ejemplo, el primer elemento de datos de usuario en la serie de caracteres se pasa a la primera salida de emisión en una secuencia de salidas de emisión.

Puede establecer la propiedad SENDEXITINIT utilizando la herramienta de administración de IBM MQ JMS o IBM MQ Explorer. Como alternativa, una aplicación puede establecer la propiedad llamando al método setSendExitInit().

Del mismo modo, la propiedad RECEXITINIT de un objeto ConnectionFactory especifica los datos de usuario que se pasan a cada salida de recepción y la propiedad SECEXITINIT especifica los datos de usuario que se pasan a una salida de seguridad. Puede establecer estas propiedades utilizando la herramienta de administración de IBM MQ JMS o IBM MQ Explorer. Como alternativa, una aplicación puede establecer esas propiedades llamando a los métodos setReceiveExitInit() y setSecurityExitInit().

Se aplican las reglas siguientes cuando se especifican datos de usuario que se pasan a salidas de canal:

- Si el número de elementos de datos de usuario en una serie de caracteres es mayor que el número de salidas en una secuencia, no se tienen en cuenta los elementos sobrantes de datos de usuario.

- Si el número de elementos de datos de usuario en una serie de caracteres es menor que el número de salidas en una secuencia, cada elemento de datos de usuario sin especificar se establece en una serie de caracteres vacía. Dos comas sucesivas en una serie de caracteres o una coma al principio de una serie de caracteres también indica un elemento de datos de usuario sin especificar.

Si una aplicación utiliza una tabla de definición de canal de cliente (CCDT) para conectar con un gestor de colas, los datos de usuario especificados en una definición de canal de conexión de cliente se pasan a las salidas de canal cuando se invocan. Si desea más información sobre la utilización de las tablas de definición de canal de cliente, consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS" en la página 292.

Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS
Una aplicación cliente de IBM MQ classes for JMS puede utilizar definiciones de canal de conexión de cliente que están almacenadas en una tabla de definición de canal de cliente (CCDT). Un objeto ConnectionFactory se configura para utilizar la CCDT. Estas son algunas de las restricciones de su uso.

Como alternativa a la creación de una definición de canal de conexión de cliente estableciendo determinadas propiedades de un objeto ConnectionFactory, una aplicación de IBM MQ classes for JMS puede utilizar las definiciones de canal de conexión de cliente que se almacenan en una tabla de definición de canal de cliente. Estas definiciones se crean mediante los mandatos IBM MQ Script (MQSC) o los mandatos IBM MQ Programmable Command Format (PCF). Cuando la aplicación crea un objeto Connection, IBM MQ classes for JMS busca en la tabla de definición de canal de cliente una definición de canal adecuada, y utiliza esa definición para iniciar un canal MQI. Para obtener más información sobre las tablas de definición de canal de cliente y sobre cómo construir una, consulte Tabla de definición de canal de cliente.

Para utilizar una tabla de definición de canal de cliente, la propiedad CCDTURL de un objeto ConnectionFactory se debe establecer en un objeto de URL. IBM MQ classes for JMS no lee la información sobre la CCDT a partir del archivo de configuración de IBM MQ MQI client, aunque algunos otros valores utilizados proceden de allí (consulte "El archivo de configuración IBM MQ classes for JMS/Jakarta Messaging" en la página 102 para conocer los valores aplicables). El objeto de URL encapsula un URL (localizador uniforme de recursos) que identifica el nombre y la ubicación del archivo que contiene la tabla de definición de canal de cliente y especifica cómo se puede acceder al archivo. Puede establecer la propiedad CCDTURL utilizando la herramienta de administración de IBM MQ JMS, o una aplicación puede establecer la propiedad creando un objeto URL y llamando al método setCCDTURL() del objeto ConnectionFactory.

Por ejemplo, si el archivo ccdt1.tab contiene una tabla de definición de canal de cliente y se almacena en el mismo sistema en el que se ejecuta la aplicación, la aplicación puede establecer la propiedad CCDTURL de esta manera:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Otro ejemplo, suponga que el archivo ccdt2.tab contiene una tabla de definición de canal de cliente y está almacenado en un sistema que es diferente del que aquél en el que se ejecuta la aplicación. Si se puede acceder al archivo utilizando el protocolo FTP, la aplicación puede establecer la propiedad CCDTURL de esta manera:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Además de establecer la propiedad CCDTURL del objeto ConnectionFactory, la propiedad QMANAGER del mismo objeto debe establecerse en uno de los valores siguientes:

- El nombre de un gestor de colas
- Un asterisco (*) seguido por el nombre de un grupo de gestores de colas.

Estos son los mismos valores que se pueden utilizar para el parámetro **QMgrName** en una llamada MQCONN emitida por una aplicación cliente que está utilizando Interfaz de Colas de Mensajes (MQI). Para

obtener más información sobre el significado de estos valores, consulte MQCONN. Puede establecer la propiedad QMANAGER utilizando la herramienta de administración de IBM MQ JMS o IBM MQ Explorer. Como alternativa, una aplicación puede establecer la propiedad llamando al método setQueueManager() del objeto ConnectionFactory.

Si una aplicación crea entonces un objeto Connection a partir del objeto ConnectionFactory, IBM MQ classes for JMS accede a la tabla de definición de canal de cliente identificada por la propiedad CCDTURL, utiliza la propiedad QMANAGER para buscar en la tabla una definición de canal de conexión de cliente adecuada y luego utiliza la definición de canal para iniciar un canal MQI para un gestor de colas.

Observe que las propiedades CCDTURL y CHANNEL de un objeto ConnectionFactory no pueden ambas estar establecidas cuando la aplicación invoca el método createConnection(). Si ambas propiedades están establecidas, el método emite una excepción. Se considera que la propiedad CCDTURL o CHANNEL está establecida si su valor es distinto de nulo, una serie vacía o una serie de caracteres en blanco.

Cuando IBM MQ classes for JMS encuentra una definición de canal de conexión de cliente adecuada en la tabla de definición de canal de cliente, sólo utiliza la información extraída de la tabla para iniciar un canal MQI. No se tiene en cuenta ninguna propiedad de canal del objeto ConnectionFactory.

En particular, tenga en cuenta lo siguiente si está utilizando TLS:

- Un canal MQI utiliza TLS sólo si la definición de canal extraída de la tabla de definición de canal de cliente especifica el nombre de una CipherSpec soportada por IBM MQ classes for JMS.
- Una tabla de definición de canal de cliente también contiene información sobre la ubicación de los servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados (listas CRL). IBM MQ classes for JMS sólo utiliza esta información para acceder a servidores LDAP que contienen listas de revocación de certificados.
- Una tabla de definición de canal de cliente también puede contener la ubicación de un programa de respuesta OCSP. IBM MQ classes for JMS no puede utilizar la información OCSP en un archivo de tabla de definición de canal de cliente. Sin embargo, puede configurar OCSP tal como se describe en la sección Online Certificate Status Protocol (OCSP) en aplicaciones cliente de Java y JMS.

Para obtener más información sobre la utilización de TLS con una tabla de definición de canal de cliente, consulte Utilización del cliente transaccional ampliado con canales TLS.

Tenga también en cuenta los puntos siguientes si está utilizando salidas de canal:

- Un canal MQI sólo utiliza las salidas de canal y los datos de usuario asociados que están especificados en la definición de canal extraída de la tabla de definición de canal de cliente.
- Una definición de canal extraída de una tabla de definición de canal de cliente puede especificar salidas de canal que están escritas en Java. Esto significa, por ejemplo, que el parámetro SCYEXIT del mandato DEFINE CHANNEL para crear una definición de canal de conexión de cliente puede especificar el nombre de una clase que implementa la interfaz WMQSecurityExit. Del mismo modo, el parámetro SENDEXIT puede especificar el nombre de una clase que implementa la interfaz WMQSendExit, y el parámetro RCVEXIT puede especificar el nombre de una clase que implementa la interfaz WMQReceiveExit. Para obtener más información sobre cómo escribir una salida de canal en Java, consulte “Escritura de salida de canal en Java para IBM MQ classes for JMS” en la página 288.

  También se pueden utilizar salidas de canal escritas en un lenguaje distinto de Java. Para obtener información sobre cómo especificar los parámetros SCYEXIT, SENDEXIT y RCVEXIT en el mandato DEFINE CHANNEL para salidas de canal escritas en otro lenguaje, consulte DEFINE CHANNEL.

Reconexión automática de cliente JMS

Configure un cliente JMS para que se vuelva a conectar de forma automática tras un fallo de red, del gestor de colas o del servidor.

Normalmente, si una aplicación IBM MQ classes for JMS autónoma está conectada a un gestor de colas utilizando el transporte de cliente y el gestor de colas deja de estar disponible por algún motivo (debido a una interrupción de red, una anomalía de gestor de colas, o el gestor de colas se está deteniendo, por ejemplo), IBM MQ classes for JMS lanzará una JMSException, la próxima vez que la aplicación intente comunicarse con el gestor de colas. La aplicación tiene que capturar la JMSException e intentar reconectarse con el gestor de colas. Se puede simplificar el diseño de la aplicación habilitando una

reconexion de cliente automática. Cuando el gestor de colas deja de estar disponible, las IBM MQ classes for JMS intentan reconectar con él de forma automática en nombre de la application. Esto significa que la aplicación no necesita contener lógica de reconexión.

El uso de esta implementación de la reconexión automática de cliente no está soportado dentro de los servidores de aplicaciones Java Platform, Enterprise Edition. Consulte "Utilización de la reconexión automática de cliente en entornos Java EE" en la página 300 para obtener una implementación alternativa.

Utilización de la reconexión automática de cliente de JMS
Si una aplicación IBM MQ classes for JMS autónoma utiliza una fábrica de conexiones que tiene la propiedad CONNECTIONNAMELIST o CCDTURL establecida, la aplicación es apta para utilizar la reconexión automática de cliente.

La reconexión automática de cliente se puede utilizar para volver a conectarse a gestores de colas, incluyendo aquellos que forman parte de una configuración de alta disponibilidad (HA). Las configuraciones de HA incluyen gestores de colas de varias instancias, gestores de colas RDQM o gestores de colas de HA en un dispositivo IBM MQ.

El comportamiento de la función de reconexión automática del cliente que proporcionan las IBM MQ classes for JMS depende de las propiedades siguientes:

La propiedad TRANSPORT de la fábrica de conexiones JMS (nombre abreviado TRAN)

TRANSPORT especifica cómo se conectan a un gestor de colas las aplicaciones que utilizan la fábrica de conexiones. Esta propiedad se debe establecer en el valor CLIENT para poder utilizar la reconexión automática del cliente. La reconexión automática del cliente no está disponible en las aplicaciones que se conectan a un gestor de colas que utiliza una fábrica de conexiones con la propiedad TRANSPORT establecida en BIND, DIRECT o DIRECTHTTP.

La propiedad QMANAGER de la fábrica de conexiones JMS (Nombre abreviado QMGR)

La propiedad QMANAGER especifica el nombre del gestor de colas al que se conecta la fábrica de conexiones.

La propiedad CONNECTIONNAMELIST de la fábrica de conexiones JMS (nombre abreviado CRHOSTS)

La propiedad CONNECTIONNAMELIST es una lista separadas por comas en la que cada entrada contiene información sobre el nombre de host y el puerto que se han de utilizar para conectarse con el gestor de colas que especifica la propiedad QMANAGER cuando se utiliza el transporte de tipo CLIENT. La lista tiene el formato siguiente: nombre de host(puerto, nombre de host(puerto).

La propiedad de fábrica de conexiones JMS CCDTURL (nombre abreviado CCDT)

La propiedad CCDTURL apunta a la tabla de definición de canal de cliente que utiliza IBM MQ classes for JMS cuando se conecta a un gestor de colas mediante CCDT.

La propiedad CLIENTRECONNECTOPTIONS de JMS (Nombre abreviado CROPT)

CLIENTRECONNECTOPTIONS controla si las IBM MQ classes for JMS intentarán la conexión automática con un gestor de colas en nombre de una aplicación, si está disponible un gestor de colas.

El atributo DefRecon de la stanza Channels del archivo de configuración del cliente

El atributo DefRecon proporciona una opción de administración que habilita la reconexión automática para todas las aplicaciones, o inhabilita la reconexión automática para las aplicaciones establecidas para la reconexión automática.

La reconexión automática del cliente solo está disponible cuando una aplicación se conecta correctamente a un gestor de colas.

Cuando una aplicación se conecta a un gestor de colas que utiliza el transporte de tipo CLIENT, las IBM MQ classes for JMS utilizan el valor de la propiedad CLIENTRECONNECTOPTIONS de la fábrica de conexiones para determinar si se utiliza la reconexión automática, cuando el gestor de colas al que está conectada la aplicación pasa a estar disponible. La tabla 1 muestra los valores posibles de la propiedad

CLIENTRECONNECTOPTIONS y el comportamiento de las IBM MQ classes for JMS para cada uno de estos valores:

Tabla 44. Valores posibles de la propiedad CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Comportamiento de las IBM MQ classes for JMS**
CUALQUIERA	Si CONNECTIONNAMELIST está establecido, utilice el valor de la propiedad CONNECTIONNAMELIST para abrir una conexión con una combinación de nombre de host y puerto y conectarse a cualquier gestor de colas. Para utilizar esta opción de reconexión automática de cliente, se debe establecer la propiedad QMANAGER en el valor predeterminado o en "*". Si CCDTURL está establecido, abra la tabla de definición de canal de cliente que se ha especificado mediante la propiedad CCDTURL, seleccione una entrada de la tabla y, después, utilice dicha entrada para iniciar un canal de conexión cliente con un gestor de colas. Para utilizar esta opción de reconexión de cliente automática, la propiedad QMANAGER se debe establecer en: • Un asterisco (*) • Un asterisco (*) seguido por el nombre de un grupo de gestores de colas. • Una serie vacía o una serie que contenga todos los caracteres en blanco
ASDEF	Utilice el valor de DefRecon para determinar si está disponible la reconexión automática del cliente.
DISABLED	No se realiza ninguna reconexión automática del cliente y se devuelve una JMSException a la aplicación.
QMGR	Especifica que el cliente se debe reconectar al mismo gestor de colas. Esta opción se debe utilizar para las soluciones de alta disponibilidad, donde la reconexión a otra instancia del mismo gestor de colas es necesaria. Si CONNECTIONNAMELIST está establecido, utilice el valor de la propiedad CONNECTIONNAMELIST para abrir una conexión con una combinación de nombre de host y puerto, y conectarse al gestor de colas especificado mediante la propiedad QMANAGER. Si CCDTURL está establecido, abra la tabla de definición de canal de cliente que se ha especificado mediante la propiedad CCDTURL, busque las entradas de la tabla que coincidan con el nombre del gestor de colas especificado mediante la propiedad QMANAGER y, después, utilice dichas entradas para iniciar una canal de conexión cliente con ese gestor de colas.

Si CONNECTIONNAMELIST está establecido, al realizar una reconexión automática de cliente, IBM MQ classes for JMS utiliza la información de la propiedad de la fábrica de conexiones CONNNECTIONNAMELIST para determinar a qué sistema se reconecta.

Inicialmente, las IBM MQ classes for JMS intentan la reconexión utilizando el nombre de host y puerto especificados en la primera entrada de CONNECTIONNAMELIST. Si se realiza una conexión, las IBM MQ classes for JMS intentan conectarse con el gestor de colas cuyo nombre se ha especificado en la propiedad QMANAGER. Si puede establecerse una conexión con el gestor de colas, IBM MQ classes for JMS vuelve a abrir todos los objetos de IBM MQ que la aplicación tenía abiertos antes de la reconexión de cliente automática y continúa ejecutándose como antes.

Si no se puede establecer una conexión con el gestor de colas necesario utilizando la primera entrada de CONNECTIONNAMELIST, las IBM MQ classes for JMS lo intentan con la segunda entrada de CONNECTIONNAMELIST, y así sucesivamente.

Cuando las IBM MQ classes for JMS han probado todas las entradas de CONNECTIONNAMELIST, esperan durante un periodo de tiempo antes de volver a intentar la reconexión. Para realizar un nuevo intento de reconexión, las IBM MQ classes for JMS comienzan por la primera entrada de CONNECTIONNAMELIST. A continuación, continúan intentando cada entrada de CONNECTIONNAMELIST por orden hasta que se lleva a cabo una reconexión o hasta llegar al final de CONNECTIONNAMELIST, donde las IBM MQ classes for JMS esperan durante un periodo de tiempo antes de volver a intentarlo.

Si CCDTURL está establecido, al realizar la reconexión automática de cliente, IBM MQ classes for JMS utiliza la tabla de definición de canal de cliente que se ha especificado en la propiedad CCDTURL para determinar a qué sistema se reconecta.

IBM MQ classes for JMS analiza inicialmente la tabla de definición de canal de cliente y encuentra una entrada adecuada que coincide con el valor de la propiedad QMANAGER. Cuando se encuentra una entrada, IBM MQ classes for JMS intenta reconectarse al gestor de colas necesario utilizando dicha entrada. Si puede establecerse una conexión con el gestor de colas, IBM MQ classes for JMS vuelve a abrir todos los objetos de IBM MQ que la aplicación tenía abiertos antes de la reconexión de cliente automática y continúa ejecutándose como antes.

Si no puede establecerse una conexión con el gestor de colas necesario, IBM MQ classes for JMS busca otra entrada adecuada en la tabla de definición de canal de cliente e intenta utilizarla, y así sucesivamente.

Cuando IBM MQ classes for JMS ha probado todas las entradas adecuadas de la tabla de definición de canal de cliente, espera un periodo de tiempo antes de intentar reconectarse. Para realizar el nuevo intento de reconexión, IBM MQ classes for JMS analiza de nuevo la tabla de definición de canal de cliente y prueba la primera entrada adecuada. A continuación, probará cada entrada adecuada en la tabla de definición de canal de cliente por turnos hasta que se produzca una reconexión o se haya intentado la última entrada adecuada en la tabla de definición de canal de cliente, momento en el que IBM MQ classes for JMS espera un periodo de tiempo antes de volver a intentarlo.

Independientemente de si se utiliza CONNECTIONNAMELIST, o CCDTURL, el proceso de la reconexión automática de cliente continúa hasta que IBM MQ classes for JMS se reconecta correctamente al gestor de colas especificado mediante la propiedad QMANAGER.

De forma predeterminada, la reconexión se lleva a cabo con los intervalos siguientes:

- El primer intento se realiza después de un retardo inicial de 1 segundo, más un intervalo aleatorio de un máximo de 250 milisegundos.
- El segundo intento se realiza a los 2 segundos, más un intervalo aleatorio de un máximo de 500 milisegundos, después de que falle el primer intento.
- El tercer intento se realiza a los 4 segundos, más un intervalo aleatorio de un máximo de 1 segundo, después de que falle el segundo intento.
- El cuarto intento se realiza a los 8 segundos, más un intervalo aleatorio de un máximo de 2 segundos, después de que falle el tercer intento.
- El quinto intento se realiza a los 16 segundos, más un intervalo aleatorio de un máximo de 4 segundos, después de que falle el cuarto intento.

- El sexto intento y todos los intentos posteriores se realiza a los 25 segundos, más un intervalo aleatorio de un máximo de 6 segundos y 250 milisegundos, después de que falle el intento anterior.

Los intentos de reconexión tienen intervalos de retardo que son parcialmente fijos y parcialmente aleatorios. Esto es así para impedir la reconexión simultánea de todas las aplicaciones de las IBM MQ classes for JMS que estaban conectadas a un gestor de colas que ya no está disponible.

Si necesita aumentar los valores predeterminados, de modo que reflejen con mayor precisión el periodo de tiempo que requiere la recuperación de un gestor de colas o la activación de un gestor de colas en espera, modifique el atributo ReconDelay en la stanza Channel del archivo de configuración del cliente. Para obtener más información, consulte la sección Stanza CHANNELS del archivo de configuración del cliente.

Una aplicación de las IBM MQ classes for JMS continuará funcionando correctamente después de la reconexión automático, en función de cómo se haya diseñado. Consulte los temas relacionados para obtener información sobre cómo diseñar aplicaciones para que utilicen la función de reconexión automática.

Códigos de razón que indican que un gestor de colas ha dejado de estar disponible
Códigos de razón que indican que un gestor de colas ha dejado de estar disponible, o que no se puede alcanzar, cuando se intenta la reconexión IBM MQ classes for JMS automática.

"Reconexión automática de cliente JMS" en la página 293 proporciona una visión general de la JMSExceptions y cómo las aplicaciones se pueden reiniciar automáticamente y la información de "Utilización de la reconexión automática de cliente de JMS" en la página 294 detalla los requisitos para la reconexión automática de cliente.

La información siguiente lista los códigos de razón IBM MQ que debería comprobar la aplicación:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

La mayoría de las JMSExceptions que se vuelven a lanzar a las aplicaciones empresariales contienen una MQException enlazada que contiene el código de razón. Para implementar la lógica de reintento para los códigos de retorno de la lista anterior, las aplicaciones empresariales deben consultar esta excepción enlazada utilizando un código similar al ejemplo siguiente:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
```

```
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

Conceptos relacionados

Clases IBM MQ para JMS

Utilización de la reconexión de cliente automática en entornos Java SE y Java EE

Puede utilizar la reconexión automática de cliente de IBM MQ para facilitar diversas soluciones de alta disponibilidad (HA) y de recuperación tras desastre (DR) dentro de un entorno Java SE y Java EE.

Están disponibles distintas soluciones de HA y DR en plataformas diferentes:

- Multi Los gestores de colas de varias instancias son instancias del mismo gestor de colas configuradas en distintos servidores (consulte Gestores de colas de varias instancias). Una instancia del gestor de colas se define como la instancia activa y otra instancia se define como la instancia en espera. Si la instancia activa falla, el gestor de colas multiinstancia se reinicia automáticamente en el servidor en espera.

  Los gestores de colas activos y en espera tienen el mismo identificador de gestor de colas (QMID). Las aplicaciones cliente de IBM MQ que se conectan a un gestor de colas multiinstancia se pueden configurar para reconectarse automáticamente a una instancia en espera de un gestor de colas utilizando la reconexión de cliente automática.

- Linux RDQM (gestor de colas de datos replicados) es una solución de alta disponibilidad que está disponible en plataformas Linux (consulte Alta disponibilidad de RDQM). Una configuración RDQM consta de tres servidores configurados en un grupo de alta disponibilidad (HA), cada uno con una instancia del gestor de colas. Una instancia es el gestor de colas en ejecución, que replica síncronamente sus datos en las otras dos instancias. Si el servidor que está ejecutando este gestor de colas falla, se inicia otra instancia del gestor de colas que tiene datos actuales con los que operar. Las tres instancias del gestor de colas comparten una dirección IP flotante, de modo que solo hay que configurar los clientes con una única dirección IP. Las aplicaciones cliente que se conectan a un gestor de colas RDQM se pueden configurar para reconectarse automáticamente a una instancia en espera de un gestor de colas utilizando la reconexión automática de cliente.

- MQ Appliance Una solución de HA también se puede proporcionar a través de un par de dispositivos IBM MQ (consulte Alta disponibilidad y Recuperación tras desastre en la documentación de IBM MQ Appliance). Un gestor de colas de HA se ejecuta en uno de los dispositivos, mientras que se duplican datos de forma síncrona en la instancia en espera del gestor de colas en el otro dispositivo. Si el dispositivo primario falla, el gestor de colas se inicia automáticamente y se ejecuta en el otro dispositivo. Las dos instancias del gestor de colas se pueden configurar para compartir una dirección IP flotante, de modo que los clientes solo tienen que configurarse con una única dirección IP. Las aplicaciones cliente que se conectan a un gestor de colas de HA en un dispositivo IBM MQ se pueden configurar para que se vuelvan a conectar automáticamente a la instancia en espera de un gestor de colas utilizando la reconexión automática de cliente.

Nota: En entornos Java EE , como por ejemplo WebSphere Application Server, la reconexión automática de cliente con especificaciones de activación utilizando la funcionalidad proporcionada por IBM MQ classes for JMS no está soportada. El adaptador de recursos IBM MQ proporciona su propio mecanismo para reconectar especificaciones de activación, si el gestor de colas al que estaba conectándose la especificación de activación deja de estar disponible. Para obtener más información, consulte “Soporte de la reconexión automática de cliente en los entornos Java EE” en la página 300.

Conceptos relacionados

Gestores de colas multiinstancia

Reconexión de cliente automática

Referencia relacionada

Alta disponibilidad en RDQM

Utilización de la reconexión automática de cliente en entornos Java SE
Las aplicaciones que utilizan las IBM MQ classes for JMS que se ejecutan en entornos Java SE pueden utilizar las funciones de reconexión automática de cliente mediante la propiedad **CLIENTRECONNECTOPTIONS** de la fábrica de conexiones.

La propiedad **CLIENTRECONNECTOPTIONS** de la fábrica de conexiones utiliza dos propiedades adicionales de la fábrica de conexiones, **CONNECTIONNAMELIST** y **CCDTURL**, para determinar cómo se ha de realizar la conexión con el servidor en el que se ejecuta el gestor de colas.

La popiedad CONNECTIONNAMELIST

La propiedad **CONNECTIONNAMELIST** es una lista separada por comas que contiene información del nombre de host y puerto que se ha de utilizar para la conexión con un gestor de colas en modo cliente. Esta propiedad se utiliza con los valores **QMANAGER** y **CHANNEL**. Cuando una aplicación utiliza la propiedad **CONNECTIONNAMELIST** para crear una conexión de cliente, las IBM MQ classes for JMS intentan conectarse con cada host siguiendo el orden de la lista. Si el primer host de gestor de colas no está disponible, las IBM MQ classes for JMS intenta conectarse con el siguiente host de la lista. Si se alcanza el final de la lista de nombres de conexiones sin crear una conexión, IBM MQ classes for JMS genera el código de razón `MQRC_QMGR_NOT_AVAILABLE` IBM MQ.

Si falla el gestor de colas al que está conectada la aplicación, cualquier aplicación que utilice **CONNECTIONNAMELIST** para conectarse con dicho gestor de colas recibe una excepción que indica que el gestor de colas no está disponible. La aplicación debe capturar la excepción y borrar cualquier recurso que esté utilizando. Para crear una conexión, la aplicación debe utilizar la fábrica de conexiones. La fábrica de conexiones vuelve a intentar la conexión con cada host, siguiendo el orden de la lista pero ahora el gestor de colas que ha fallado no está disponible. La fábrica de conexiones intenta conectarse con otro host de la lista.

La popiedad CCDTURL

La propiedad **CCDTURL** contiene un URL (Uniform Resource Locator) que apunta a la tabla CCDT (Client Channel Definition Table), esta propiedad se utiliza con la propiedad **QMANAGER**. La CCDT contiene una lista de los canales de cliente que se utilizan para conectarse con un gestor de colas definido en un sistema IBM MQ. Para obtener información sobre cómo las IBM MQ classes for JMS utilizan las CCDT, consulte la sección “Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for JMS” en la página 292.

Utilización de la propiedad CLIENTRECONNECTOPTIONS para habilitar la reconexión automática de cliente en las IBM MQ classes for JMS

La propiedad **CLIENTRECONNECTOPTIONS** se utiliza para habilitar la reconexión automática de cliente en las IBM MQ classes for JMS. Los valores posibles de esta propiedad son los siguientes:

ASDEF
El comportamiento de la reconexión automática de cliente se define mediante el valor especificado en la stanza Channel del archivo de configuración de cliente de IBM MQ (`mqclient.ini`).

Inhabilitado
La reconexión automática de cliente está inhabilitada.

QMGR
Las IBM MQ classes for JMS intentan conectarse a un gestor de colas con el mismo identificador de gestor de colas que el gestor de colas al que se han conectado, utilizando cualquiera de las opciones siguientes:

- La propiedad **CONNECTIONNAMELIST** y el canal definido en la propiedad **CHANNEL**.
- La CCDT definida en la propiedad **CCDTURL**.

ANY
Las IBM MQ classes for JMS intentan la reconexión con un gestor de colas con el mismo nombre utilizando la propiedad **CONNECTIONNAMELIST** o **CCDTURL**.

Información relacionada

Stanza CHANNELS del archivo de configuración de cliente

Utilización de la reconexión automática de cliente en entornos Java EE

El adaptador de recursos IBM MQ, que se puede desplegar en entornos Java EE (Java Platform, Enterprise Edition) y el proveedor de mensajería WebSphere Application Server IBM MQ utilizan IBM MQ classes for JMS para comunicarse con gestores de colas IBM MQ. El adaptador de recursos de IBM MQ y el proveedor de mensajería de WebSphere Application Server IBM MQ proporcionan una serie de mecanismos para permitir que las especificaciones de activación, los puertos de escucha de WebSphere Application Server y las aplicaciones que se ejecutan dentro de contenedores de cliente se reconecten automáticamente a un gestor de colas. Los EJB (Enterprise JavaBeans) y las aplicaciones basadas en web deben implementar su propia lógica de reconexión.

Nota: La reconexión automática de cliente con especificaciones de activación utilizando la funcionalidad proporcionada por IBM MQ classes for JMS no está soportada (consulte "Reconexión automática de cliente JMS" en la página 293). El adaptador de recursos IBM MQ proporciona su propio mecanismo para reconectar especificaciones de activación, si el gestor de colas al que estaba conectándose la especificación de activación deja de estar disponible.

El mecanismo que proporciona el adaptador de recursos está controlado por:

- La propiedad del adaptador de recursos IBM MQ **reconnectionRetryCount**.
- La propiedad del adaptador de recursos IBM MQ **reconnectionRetryInterval**.
- La propiedad de especificación de activación **connectionNameList**.

Para obtener más información sobre estas propiedades, consulte "Configuración de las propiedades del objeto ResourceAdapter" en la página 463.

El uso de reconexiones de cliente automáticas no está soportado en el método onMessage() de una aplicación de beans controlados por mensaje, ni en ninguna otra aplicación que ejecute en un entorno Java Platform, Enterprise Edition. La aplicación tendrá que implementar su propia lógica de reconexión si el gestor de colas con que estaba conectada deja de estar disponible. Para obtener más información, consulte "Implementación de la lógica de reconexión en una aplicación Java EE" en la página 308.

Soporte de la reconexión automática de cliente en los entornos Java EE

En entornos Java EE , como por ejemplo WebSphere Application Server, el adaptador de recursos de IBM MQ y el proveedor de mensajería de WebSphere Application Server IBM MQ proporcionan una serie de mecanismos que permiten que las aplicaciones se reconecten a un gestor de colas automáticamente. No obstante, en algunos casos, se aplican restricciones a este soporte.

El adaptador de recursos IBM MQ que se puede desplegar en entornos Java EE y el proveedor de mensajería WebSphere Application Server IBM MQ utilizan IBM MQ classes for JMS para comunicarse con los gestores de colas IBM MQ.

La tabla siguiente resume el soporte que proporcionan el adaptador de recursos IBM MQ y el proveedor de mensajería WebSphere Application Server IBM MQ para la reconexión automática del cliente.

Tabla 45. Resumen de soporte de las opciones de reconexión automática de cliente en los entornos de Java EE

Opciones de reconexion automática	**Propiedad CONNECTIONNAMELIST**	**Propiedad CCDTURL**	**Propiedad CLIENTRECONNECTOPTIONS**	**Método alternativo para la reconexión automática de cliente**
Especificaciones de activación	Soportada con restricciones	Soportada con restricciones	No soportado	El entorno de Java EE y las especificaciones de activación proporcionan su propio mecanismo de reconexión
Puertos de escucha de WebSphere Application Server	Soportada con restricciones	Soportada con restricciones	No soportado	WebSphere Application Server proporciona su propio mecanismo de reconexión
Aplicaciones de Enterprise JavaBeans y basadas en web.	Soportada con restricciones	Soportada con restricciones	No soportado	Las aplicaciones deben implementar su propia lógica de reconexión
Aplicaciones que ejecutan en contenedores cliente.	Soportado	Soportado	Soportado	No aplicable

Las aplicaciones de beans controlados por mensajes que están instaladas en un entorno Java EE , como por ejemplo IBM MQ classes for JMS, pueden utilizar especificaciones de activación para procesar mensajes en un sistema IBM MQ . Las especificaciones de activación se utilizan para detectar mensajes que llegan a un sistema IBM MQ y entregarlos a beans controlados por mensajes para su proceso. Los beans controlados por mensajes también pueden realizar conexiones adicionales con sistemas IBM MQ desde su método **onMessage()** interno. Para obtener más información sobre estas conexiones pueden utilizar la reconexión automática de cliente, consulte la sección Aplicaciones de Enterprise JavaBeans y basadas en web.

Especificaciones de activación
Para las especificaciones de activación, las propiedades **CONNECTIONNAMELIST** y **CCDTURL** están soportadas con restricciones y la propiedad **CLIENTRECONNECTOPTIONS** no está soportada.

Las aplicaciones de beans controlados por mensajes (MDB) que están instaladas en un entorno Java EE , como por ejemplo WebSphere Application Server, pueden utilizar especificaciones de activación para procesar mensajes en un sistema IBM MQ .

Las especificaciones de activación se utilizan para detectar los mensajes que llegan a un sistema IBM MQ y, después, entregarlos a los MDB para su proceso. En esta sección trata de cómo la especificación de activación supervisa el sistema IBM MQ.

Los MDB también pueden realizar conexiones adicionales con sistemas IBM MQ desde su método `onMessage()`.

En “Aplicaciones de Enterprise JavaBeans y basadas en web.” en la página 305 se pueden encontrar detalles sobre cómo estas conexiones pueden utilizar la reconexión automática de cliente.

La popiedad CONNECTIONNAMELIST

Cuando se inicia, la especificación de activación intenta conectar con el gestor de colas usando:

- El gestor especificado en la propiedad **QMANAGER**.
- El canal mencionado en la propiedad **CHANNEL**.
- Nombre de host e información de puerto de la primera entrada de **CONNECTIONNAMELIST**

Si la especificación de activación no puede conectarse con el gestor de colas utilizando la primera entrada de la lista, pasará a la segunda entrada y así sucesivamente hasta que se haya realizado una conexión con el gestor de colas o se haya alcanzado el final de la lista.

Si la especificación de activación no puede conectarse con el gestor de colas especificado utilizando ninguna de las entradas de **CONNECTIONNAMELIST**, dicha especificación se parará y tendrá que reiniciarse.

Una vez que se está ejecutando la especificación de activación, la especificación de activación obtiene mensajes del sistema IBM MQ y entrega los mensajes a un MDB para su proceso.

Si el gestor de colas falla mientras se está procesando un mensaje, el entorno Java EE detecta la anomalía e intenta volver a conectar la especificación de activación.

La especificación de activación utiliza la información de la propiedad **CONNECTIONNAMELIST** como antes, cuando la especificación de activación realiza intenta reconectar.

Si la especificación de activación intenta todas las entradas de **CONNECTIONNAMELIST** y sigue sin poder conectarse al gestor de colas, la especificación de activación espera un periodo de tiempo especificado por la propiedad de adaptador de recursos IBM MQ **reconnectionRetryInterval** antes de volver a intentarlo.

La propiedad del adaptador de recursos de IBM MQ **reconnectionRetryCount** define el número de intentos de reconexión consecutivos que se van a realizar antes de que se detenga una especificación de activación y requiere un reinicio manual.

Una vez que la especificación de activación se haya reconectado a un sistema IBM MQ, el entorno Java EE realiza la limpieza transaccional necesaria y reanuda la entrega de mensajes a los MDB para su proceso.

Para que la limpieza transaccional pueda funcionar correctamente, el entorno Java EE debe poder acceder a los registros para el gestor de colas que ha fallado.

Si las especificaciones de activación se están utilizando con MDB transaccionales que participan en transacciones XA y se están conectando con un gestor de colas multiinstancia, la **CONNECTIONNAMELIST** habrá de contener una entrada para las instancias de gestor de colas activa y en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si los MDB transaccionales se están utilizando con gestores de colas autónomos, la propiedad **CONNECTIONNAMELIST** debe contener una sola entrada, para asegurarse de que la especificación de activación siempre se vuelva a conectar al mismo gestor de colas que se ejecuta en el mismo sistema después de una anomalía.

La popiedad CCDTURL

Cuando se inicia, la especificación de activación intenta conectarse al gestor de colas especificado en la propiedad **QMANAGER** utilizando la primera entrada de la tabla de definiciones de canal de cliente (CCDT).

Si la especificación de activación no puede conectarse con el gestor de colas utilizando la primera entrada de la tabla, pasará a la segunda entrada y así sucesivamente hasta que se haya realizado una conexión con el gestor de colas o se haya alcanzado el final de la tabla.

Si la especificación de activación no puede conectarse con el gestor de colas especificado utilizando ninguna de las entradas de la CCDT, dicha especificación se parará y tendrá que reiniciarse.

Una vez que se está ejecutando la especificación de activación, la especificación de activación obtiene mensajes del sistema IBM MQ y entrega los mensajes a un MDB para su proceso.

Si el gestor de colas falla mientras se está procesando un mensaje, el entorno Java EE detecta la anomalía e intenta volver a conectar la especificación de activación.

La especificación de activación utiliza la información de la propiedad CCDT como antes, cuando la especificación de activación realiza intenta reconectar.

Si la especificación de activación intenta todas las entradas de CCDT y sigue sin poder conectarse al gestor de colas, la especificación de activación espera un periodo de tiempo especificado por la propiedad del adaptador de recursos de IBM MQ **reconnectionRetryInterval** antes de volver a intentarlo.

La propiedad del adaptador de recursos de IBM MQ **reconnectionRetryCount** define el número de intentos de reconexión consecutivos que se van a realizar antes de que se detenga una especificación de activación y requiere un reinicio manual.

Una vez que la especificación de activación se haya reconectado a un sistema IBM MQ, el entorno Java EE realiza la limpieza transaccional necesaria y reanuda la entrega de mensajes a los MDB para su proceso.

Para que la limpieza transaccional pueda funcionar correctamente, el entorno Java EE debe poder acceder a los registros para el gestor de colas que ha fallado.

Si las especificaciones de activación se están utilizando con MDB transaccionales que participan en transacciones XA y se están conectando con un gestor de colas multiinstancia, la CCDT habrá de contener una entrada para las instancias de gestor de colas activa y en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si los MDB transaccionales se utilizan con gestores de colas autónomos, la CCDT habrá de contener una única entrada para asegurarse de que la especificación de activación siempre se reconecte con el mismo gestor de colas que ejecuta en el mismo sistema tras un error.

Asegúrese de haber establecido el valor predeterminado *PREFERRED* en la propiedad **AFFINITY** de las CCDT que se utiliza con las especificaciones de activación, para que las conexiones se realicen en el mismo gestor de colas activo.

La popiedad CLIENTRECONNECTOPTIONS

Las especificaciones de activación proporcionan su propia funcionalidad de reconexión. La funcionalidad proporcionada permite que las especificaciones se reconecten automáticamente a un sistema IBM MQ, si el gestor de colas al que estaban conectadas falla.

Por ello, no se da soporte a la funcionalidad de reconexión automática de cliente proporcionada por IBM MQ classes for JMS.

Debe establecer la propiedad **CLIENTRECONNECTOPTIONS** en *DISABLED* para todas las especificaciones de activación que se utilizan en Java EE.

Puertos de escucha de WebSphere Application Server
Las de beans controlados por mensajes (MDB) instaladas en WebSphere Application Server también pueden utilizar los puertos de escucha para procesar mensajes en el sistema IBM MQ.

Los puertos de escucha se utilizan para detectar los mensajes que llegan a un sistema IBM MQ y, a continuación, entregarlos a los MDB para su proceso. En este tema se describe cómo el puerto de escucha supervisa el sistema IBM MQ.

Los MDB también pueden realizar conexiones adicionales con sistemas IBM MQ desde su método `onMessage()`.

Consulte la sección “Aplicaciones de Enterprise JavaBeans y basadas en web.” en la página 305 para obtener más información sobre cómo estas conexiones pueden utilizar la reconexión automática de cliente.

Para los puertos de escucha de WebSphere Application Server:

- Se da soporte a **CONNECTIONNAMELIST** y **CCDTURL** con restricciones
- **CLIENTRECONNECTOPTIONS** no está soportado

La popiedad CONNECTIONNAMELIST

Los puertos de escucha utilizan las agrupaciones de conexiones de JMS cuando se conectan a IBM MQ, por lo tanto, están sujetas a las implicaciones del uso de agrupaciones de conexiones. En “Especificaciones de activación” en la página 301 encontrará más información.

Si no hay ninguna conexión libre y todavía no se ha creado el número máximo de conexiones de esta fábrica de conexiones, se utiliza la propiedad **CONNECTIONNAMELIST** para intentar y crear una nueva conexión con IBM MQ.

Si todos los sistemas IBM MQ que figuran en **CONNECTIONNAMELIST** no están accesibles, el puerto de escucha se detiene.

A continuación, el puerto de escucha espera durante el periodo de tiempo especificado en la propiedad personalizada **RECOVERY.RETRY.INTERVAL** del servicio de escucha de mensajes y vuelve a intentar la reconexión.

Este intento de reconexión comprueba si hay alguna conexión libre en la agrupación de conexiones, por si se ha devuelto alguna entre los intentos de conexión. Si no hay ninguna disponible, el puerto de escucha utiliza **CONNECTIONNAMELIST**, como antes.

Cuando el puerto de escucha se ha reconectado con un sistema IBM MQ, el entorno Java EE realiza cualquier limpieza de transacciones necesaria y, a continuación, reanuda la entrega de mensajes a los MDB para su proceso.

Para que la limpieza transaccional pueda funcionar correctamente, el entorno Java EE debe poder acceder a los registros para el gestor de colas que ha fallado.

Si se están utilizando los puertos de escucha con los MDB transaccionales que participan en transacciones XA y se conectan a un **gestor de colas de varias instancias**, **CONNECTIONNAMELIST** debe contener una entrada para la instancia del gestor de colas activo y para el que está en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si los MDB transaccionales se están utilizando con gestores de colas autónomos, la propiedad **CONNECTIONNAMELIST** debe contener una sola entrada, para asegurarse de que la especificación de activación siempre se vuelva a conectar al mismo gestor de colas que se ejecuta en el mismo sistema después de una anomalía.

La popiedad CCDTURL

Cuando se inicia, el puerto de escucha intenta conectarse al gestor de colas especificado en la propiedad **QMANAGER** utilizando la primera entrada de la CCDT.

Si el puerto de escucha no se puede conectar con el gestor de colas mediante la primera entrada de la tabla, el puerto de escucha pasa a la segunda entrada y así sucesivamente, hasta que se realiza una conexión con el gestor de colas o hasta que se alcanza el final de la tabla.

Si puerto de escucha no se puede conectar con el gestor de colas especificado utilizando cualquiera de las entradas de CCDT, el puerto de escucha se detiene.

A continuación, el puerto de escucha espera durante el periodo de tiempo especificado en la propiedad personalizada **RECOVERY.RETRY.INTERVAL** del servicio de escucha de mensajes y vuelve a intentar la reconexión.

Este intento de reconexión recorre todas las entradas de CCDT, como en el caso anterior.

Cuando el puerto de escucha se está ejecutando, obtiene los mensajes del sistema IBM MQ y los entrega a un MDB para su proceso.

Si el gestor de colas falla mientras se está procesando un mensaje, el entorno Java EE detecta el error e intenta reconectar el puerto de escucha. El puerto de escucha utiliza la información de CCDT cuando realiza los intentos de reconexión.

Si el puerto de escucha prueba todas las entradas de CCDT y sigue sin poder conectarse al gestor de colas, espera durante el periodo de tiempo especificado en la propiedad **RECOVERY.RETRY.INTERVAL** antes de volver a intentarlo.

La propiedad **MAX.RECOVERY.RETRIES** del servicio de escucha de mensajes define el número de intentos de reconexión consecutivos que se pueden realizar antes de que se detenga el puerto de escucha y éste requiera un reinicio manual.

Cuando el puerto de escucha se ha reconectado con un sistema IBM MQ, el entorno Java EE realiza cualquier limpieza de transacciones necesaria y, a continuación, reanuda la entrega de mensajes a los MDB para su proceso.

Para que la limpieza transaccional pueda funcionar correctamente, el entorno Java EE debe poder acceder a los registros para el gestor de colas que ha fallado.

Si se están utilizando los puertos de escucha con los MDB transaccionales que participan en transacciones XA y se conectan a un gestor de colas de varias instancias, CCDT debe contener una entrada para la instancia del gestor de colas activo y para el que está en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si los MDB transaccionales se están utilizando con los gestores de colas autónomos, CCDT debe contener una sola entrada para asegurarse de que el puerto de escucha siempre se reconecte con el mismo gestor de colas que se ejecuta en el mismo sistema, en caso de que se produzca una anomalía.

Asegúrese de que ha establecido el valor *PREFERRED* para la propiedad **AFFINITY** en las CCDT utilizadas por los puertos de escucha, de modo que se realicen las conexiones con el mismo gestor de colas activo.

La popiedad CLIENTRECONNECTOPTIONS

Los puertos de escucha proporcionan sus propias funciones de reconexión. Las funciones proporcionadas permiten que los puertos de escucha se reconecten automáticamente con un sistema IBM MQ si falla el gestor de colas al que estaban conectados.

Por ello, no se da soporte a la funcionalidad de reconexión automática de cliente proporcionada por IBM MQ classes for JMS.

Debe establecer la propiedad **CLIENTRECONNECTOPTIONS** en *DISABLED* para todos los puertos de escucha que se utilizan en Java EE.

Aplicaciones de Enterprise JavaBeans y basadas en web.
Las aplicaciones de Enterprise JavaBean (EJB) y las aplicaciones que se ejecutan en un contenedor web como, por ejemplo, los servlets, utilizan una fábrica de conexiones JMS para crear una conexión con un gestor de colas de IBM MQ.

Se aplican las restricciones siguientes a las aplicaciones EJB y basadas en web:

- Se da soporte a **CONNECTIONNAMELIST** y **CCDTURL** con restricciones
- **CLIENTRECONNECTOPTIONS** no está soportado

La popiedad CONNECTIONNAMELIST

Si el entorno de Java EE proporciona una agrupación de conexiones para las conexiones JMS, consulte "Utilización de CONNECTIONNAMELIST o CCDT en una agrupación de conexiones" en la página 307 para obtener información sobre cómo afecta al comportamiento de la propiedad **CONNECTIONNAMELIST**.

Si el entorno Java EE no proporciona una agrupación de conexiones JMS, la aplicación utiliza la propiedad **CONNECTIONNAMELIST** de la misma forma que las aplicaciones Java SE.

Si las aplicaciones se utilizan con MDB transaccionales que participan en transacciones XA y se están conectando a un gestor de colas multiinstancia, **CONNECTIONNAMELIST** debe contener una entrada para las instancias del gestor de colas activo y en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si las aplicaciones se están utilizando con gestores de colas autónomos, la propiedad **CONNECTIONNAMELIST** debe contener una sola entrada, para asegurarse de que la aplicación siempre se vuelva a conectar al mismo gestor de colas, que se ejecuta en el mismo sistema, después de una anomalía.

La popiedad CCDTURL

Si el entorno de Java EE proporciona una agrupación de conexiones para las conexiones JMS, consulte "Utilización de CONNECTIONNAMELIST o CCDT en una agrupación de conexiones" en la página 307 para obtener información sobre cómo afecta al comportamiento de la propiedad **CCDTURL**.

Si el entorno Java EE no proporciona una agrupación de conexiones JMS, la aplicación utiliza la propiedad **CCDTURL** de la misma forma que las aplicaciones Java SE.

Si las aplicaciones se utilizan con MDB transaccionales que participan en transacciones XA y se están conectando a un gestor de colas multiinstancia, CCDT debe contener una entrada para las instancias del gestor de colas activo y en espera.

Esto significa que el entorno Java EE puede acceder a los registros del gestor de colas, si el entorno debe realizar la recuperación de transacciones, independientemente del gestor de colas al que se reconecte el entorno después de una anomalía.

Si las aplicaciones se utilizan con gestores de colas autónomos, CCDT debe contener una sola entrada, para asegurarse de que la especificación de activación siempre se vuelva a conectar al mismo gestor de colas, que se ejecuta en el mismo sistema, después un error.

La popiedad CLIENTRECONNECTOPTIONS

Debe establecer la propiedad **CLIENTRECONNECTOPTIONS** en *DISABLED* para todas las fábricas de conexiones JMS utilizadas por los EJB o las aplicaciones que se ejecutan en el contenedor web.

Las aplicaciones que deben volver a conectarse automáticamente a un nuevo gestor de colas si el gestor de colas que están utilizando falla tienen que implementar su propia lógica de reconexión. Consulte "Implementación de la lógica de reconexión en una aplicación Java EE" en la página 308 para obtener más información.

Escenarios: WebSphere Application Server con IBM MQ
Escenarios: perfil de Liberty de WebSphere Application Server con IBM MQ

Aplicaciones que ejecutan en contenedores cliente.
Algunos entornos de Java EE como, por ejemplo, WebSphere Application Server, proporcionan un contenedor de cliente que se puede utilizar para ejecutar aplicaciones Java SE.

Las aplicaciones que se ejecutan dentro de estos entornos utiliza una fábrica de conexiones JMS para conectar con un gestor de colas de IBM MQ.

Para las aplicaciones que se ejecutan dentro de contenedores de cliente:

- **CONNECTIONNAMELIST** y **CCDTURL** están totalmente soportados
- **CLIENTRECONNECTOPTIONS** está totalmente soportado

La popiedad CONNECTIONNAMELIST

Si el entorno de Java EE proporciona una agrupación de conexiones para las conexiones JMS, consulte "Utilización de CONNECTIONNAMELIST o CCDT en una agrupación de conexiones" en la página 307 para obtener información sobre cómo afecta al comportamiento de la propiedad **CONNECTIONNAMELIST**.

Si el entorno Java EE no proporciona una agrupación de conexiones JMS, la aplicación utiliza la propiedad **CONNECTIONNAMELIST** de la misma forma que las aplicaciones Java SE.

La popiedad CCDTURL

Si el entorno de Java EE proporciona una agrupación de conexiones para las conexiones JMS, consulte "Utilización de CONNECTIONNAMELIST o CCDT en una agrupación de conexiones" en la página 307 para obtener información sobre cómo afecta al comportamiento de la propiedad **CCDTURL**.

Si el entorno Java EE no proporciona una agrupación de conexiones JMS, la aplicación utiliza la propiedad **CCDTURL** de la misma forma que las aplicaciones Java SE.

Utilización de CONNECTIONNAMELIST o CCDT en una agrupación de conexiones
Algunos entornos Java EE, por ejemplo, WebSphere Application Server, proporcionan una agrupación de conexiones JMS. Contenedor que se puede utilizar para ejecutar aplicaciones Java SE.

Las aplicaciones que crean una conexión utilizando una fábrica de conexiones que se ha definido en el entorno Java EE obtienen una conexión libre existente de la agrupación de conexiones para esta fábrica de conexiones, o bien una nueva conexión si no hay ninguna adecuada en la agrupación de conexiones.

Esto puede tener implicaciones si la fábrica de conexiones se ha configurado definiendo la propiedad **CONNECTIONNAMELIST** o la propiedad **CCDTURL**.

La primera vez que se utiliza la fábrica de conexiones para crear una conexión, el entorno Java EE utiliza **CONNECTIONNAMELIST** o **CCDTURL** para crear una nueva conexión al sistema IBM MQ. Cuando esta conexión ya no es necesaria, se devuelve a la agrupación de conexiones, donde pasa a estar disponible para su reutilización.

Si algo más crea una conexión desde la fábrica de conexiones, el entorno Java EE devuelve la conexión de la agrupación de conexiones, en lugar de utilizar las propiedades **CONNECTIONNAMELIST** o **CCDTURL** para crear una nueva conexión.

Si se está usando una conexión cuando falla una instancia del gestor de colas, dicha conexión se descarta. Sin embargo, puede que no se descarte el contenido de la agrupación de conexiones, lo que significaría que la agrupación podría potencialmente contener conexiones con un gestor de colas que ya no está ejecutando.

En esta situación, la próxima vez que se realice una petición para crear una conexión desde la fábrica de conexiones, se devolverá una conexión con el gestor de colas fallido. Cualquier intento de utilizar esta conexión fallará, ya que el gestor de colas ya no está ejecutando, lo que hace que la conexión se descarte.

Solo cuando la agrupación de conexiones esté vacía, el entorno Java EE usará las propiedades **CONNECTIONNAMELIST** o **CCDTURL** para crear una conexión con IBM MQ.

Debido a la forma en la que se utilizan **CONNECTIONNAMELIST** y CCDT para crear conexiones JMS, también es posible tener una agrupación de conexiones que contenga conexiones con sistemas IBM MQ diferentes.

Por ejemplo, suponga que se ha configurado una fábrica de conexiones con la propiedad **CONNECTIONNAMELIST** establecida al valor siguiente:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Supongamos que la primera vez que una aplicación intenta crear una conexión con un gestor de colas autónomo desde esta fábrica de conexiones, el gestor de colas que se ejecuta en el sistema `hostname1(port1)` no es accesible. Esto significa que la aplicación termina con una conexión con el gestor de colas que se ejecuta en `hostname2(port2)`.

Ahora aparece otra aplicación y crea una conexión JMS desde la misma fábrica de conexiones. Ahora el gestor de colas en `hostname1(port1)` está disponible, así que se crea una nueva conexión JMS con este sistema IBM MQ y se devuelve a la aplicación.

Cuando ambas aplicaciones han finalizado, cierran sus conexiones JMS, lo que hace que las conexiones se devuelvan a la agrupación de conexiones.

El resultado es que la agrupación de conexiones para nuestra fábrica de conexiones ahora contiene dos conexiones JMS:

- Una conexión con el gestor de colas que se ejecuta en `hostname1(port1)`
- Una conexión con el gestor de colas que se ejecuta en `hostname2(port2)`

Esto puede dar lugar a problemas relacionados con la recuperación de transacciones. Si el sistema Java EE necesita retrotraer una transacción, debe poder conectarse a un gestor de colas que tenga acceso a los registros de transacciones.

Implementación de la lógica de reconexión en una aplicación Java EE
Enterprise JavaBeans y aplicaciones basadas en web que desean reconectarse automáticamente si un gestor de colas no puede implementar su propia lógica de reconexión.

Las siguientes opciones proporcionan más información sobre cómo puede conseguirlo.

Dejar que la aplicación falle

Este enfoque no requiere ningún cambio de aplicación, pero sí una reconfiguración administrativa de la definición de fábrica de conexiones para incluir la propiedad **CONNECTIONNAMELIST**. No obstante, este enfoque requiere que el invocador pueda manejar un error correctamente. Tenga en cuenta que esto también es necesario para errores como MQRC_Q_FULL, que no están relacionados con ningún error de conexión.

Código de ejemplo para este proceso:

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
          try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

En el código anterior se supone que la fábrica de conexiones que utiliza este servlet tiene definida la propiedad **CONNECTIONNAMELIST**.

Cuando el servlet se procesa por primera vez, se crea una nueva conexión utilizando la propiedad **CONNECTIONNAMELIST** , suponiendo que no haya conexiones agrupadas disponibles desde otras aplicaciones que se conecten al mismo gestor de colas.

Cuando se libera la conexión después de una llamada `close()`, la conexión se devuelve a la agrupación y se reutiliza la próxima vez que se ejecute el servlet, sin hacer referencia a **CONNECTIONNAMELIST**, hasta que se produzca un error de conexión y se genere un suceso CONNECTION_ERROR_OCCURRED. Este suceso solicita a la agrupación que destruya la conexión fallida.

Cuando se ejecuta la aplicación, no hay disponible ninguna conexión agrupada y se utiliza **CONNECTIONNAMELIST** para conectarse al primer gestor de colas disponible. Si se ha producido una migración tras error del gestor de colas (por ejemplo, el error no ha sido un error de red transitorio), el servlet se conecta a la instancia de copia de seguridad cuando esté disponible.

Si hay otros recursos implicados en la aplicación como, por ejemplo, bases de datos, puede que sea adecuado indicar que el servidor de aplicaciones deberá retrotraer la transacción.

Manejar la reconexión dentro de la aplicación

Si el invocador no puede procesar un error del servlet, la reconexión debe manejarse dentro de la aplicación. Como se muestra en el ejemplo siguiente, para manejar una reconexión dentro de la aplicación, la aplicación debe solicitar una nueva conexión para que pueda almacenar en memoria caché la fábrica de conexiones que ha consultado en JNDI y manejar una `JMSException` como, por ejemplo, `JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN')`.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

En el ejemplo siguiente, `setupResources()` crea los objetos JMS e incluye un bucle de inactividad y reintento para manejar la reconexión no instantánea. En la práctica, este método evita muchos intentos de reconexión. Tenga en cuenta que las condiciones de salida se han omitido en el ejemplo para que resulte más claro.

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
```

```
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

Si la aplicación gestiona la reconexión, es importante que la aplicación libere cualquier conexión que se retenga para otros recursos, tanto si son otros gestores de colas de IBM MQ como si son otros servicios de programa de fondo como, por ejemplo, bases de datos. Debe volver a establecer estas conexiones cuando finalice la reconexión a una nueva instancia del gestor de colas de IBM MQ. Si no vuelve a establecer las conexiones, los recursos del servidor de aplicaciones se retienen innecesariamente durante el intento de reconexión, y puede que hayan excedido el tiempo de espera en el momento en el que se reutilicen.

Uso del gestor de trabajo

Para las aplicaciones de larga duración (por ejemplo, los proceso por lotes) donde el tiempo de proceso es mayor que unas pocas decenas de segundos, se puede utilizar el gestor de trabajo de WebSphere Application Server. A continuación, se muestra un ejemplo de fragmento de código para WebSphere Application Server:

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

donde `web.xml` contiene:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

y el lote se implementa ahora mediante la interfaz de trabajo:

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");
```

```
    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Si el proceso por lotes tarda mucho tiempo en ejecutarse, por ejemplo, si los mensajes son grandes, la red va lenta o el acceso a la base de datos es muy amplio (especialmente cuando se combina con una migración tras error lenta), el servidor empieza a emitir avisos de hebra colgada parecidos al ejemplo siguiente:

WSVR0605W: La hebra "`WorkManager.DefaultWorkManager` : 0" (00000035) ha estado activa 694061 milisegundos y es posible que se haya colgado. Hay una o varias hebras en el servidor que pueden estar colgadas.

Estos avisos pueden minimizarse reduciendo el tamaño del lote o incrementando el tiempo de espera de las hebras colgadas. No obstante, generalmente es preferible implementar este proceso en un proceso de EJB (para el envío por lotes) o beans controlados por mensaje (para el consumo o el consumo y la respuesta).

Tenga en cuenta que la reconexión gestionada por la aplicación no proporciona una solución general para manejar los errores de tiempo de ejecución, y la aplicación debe seguir manejando errores que no están relacionados con el error de conexión.

Por ejemplo, cuando intenta poner un mensaje en una cola que esté llena (2053 MQRC_Q_FULL) o cuando intenta conectarse a un gestor de colas utilizando credenciales de seguridad que no son válidas (2035 MQRC_NOT_AUTHORIZED).

La aplicación también debe manejar los errores 2059 MQRC_Q_MGR_NOT_AVAILABLE cuando no hay instancias disponibles de forma inmediata cuando la migración tras error está en curso. Esto puede conseguirse si la aplicación notifica las excepciones JMS a medida que se producen, en lugar de intentar reconectarse de forma silenciosa.

Agrupación de objetos de IBM MQ classes for JMS
El uso de una forma de agrupación de conexiones fuera de Java EE ayuda a reducir la carga general resultante, por ejemplo, de algunas aplicaciones autónomas que utilizan infraestructuras, o que se despliegan en entornos de nube y, también, de un número mayor de conexiones de cliente en `QueueManagers`, lo que lleva a un aumento en la consolidación del servidor de aplicaciones y gestores de colas.

En el modelo de programación de Java EE, existe un ciclo de vida bien definido de los distintos objetos en uso. Los beans controlados por mensajes (MDB) son los más restringidos, mientras que los servlets proporcionan más libertad. Por lo tanto, las opciones de agrupación que están disponibles en los servidores Java EE se adaptan a los diferentes modelos de programación utilizados.

Con Java SE (o con otra infraestructura, como Spring) los modelos de programación son extremadamente flexibles. Por tanto, una única estrategia de agrupamientos no valdría para todos. Hay que tener en cuenta si va a haber un entorno que permita alguna forma de agrupamiento como, por ejemplo, Spring.

La estrategia de agrupación que se va a utilizar depende el entorno en el cual se está ejecutando la aplicación.

Agrupaciones de objetos en un entorno Java EE
Los servidores de aplicaciones Java EE proporcionan la funcionalidad de agrupación de conexiones que pueden utilizar las aplicaciones de beans controlados por mensajes, Enterprise Java Beans y Servlets.

WebSphere Application Server mantiene una agrupación de conexiones con un proveedor JMS para mejorar el rendimiento. Cuando una aplicación crea una conexión JMS, el servidor de aplicaciones determina si ya existe una conexión en la agrupación de conexiones libres. Si es así, se devuelve dicha conexión a la aplicación; en caso contrario, se crea una conexión.

Figura 41 en la página 312 muestra cómo tanto las especificaciones de activación como los puertos de escucha establecen una conexión JMS y utilizan dicha conexión para supervisar un destino para los mensajes en modalidad normal.

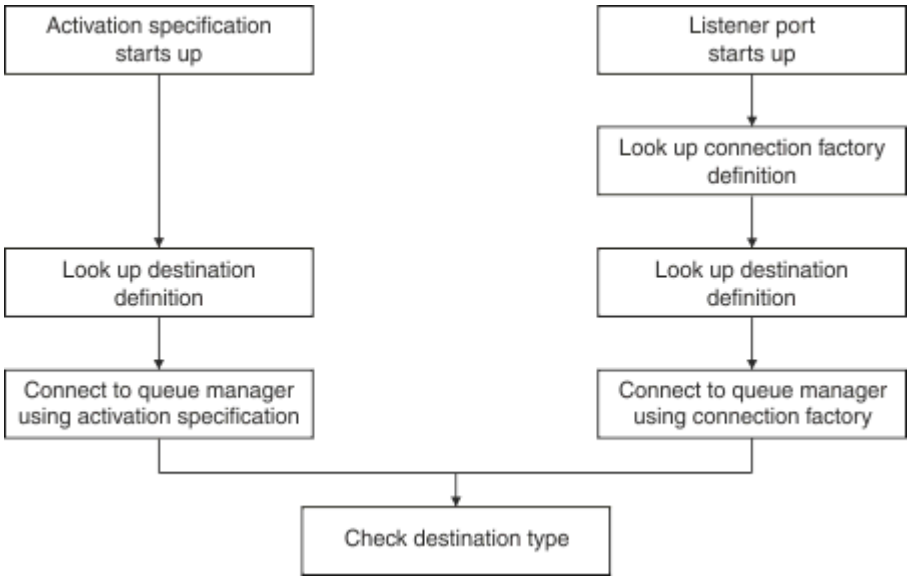


Figura 41. Modo normal

Cuando se utiliza el proveedor de mensajería de IBM MQ , las aplicaciones que realizan la mensajería de salida (por ejemplo, Enterprise Java Beans y servlets), y el componente de puerto de escucha de bean controlado por mensajes, pueden utilizar estas agrupaciones de conexiones.

Las especificaciones de activación del proveedor de mensajería IBM MQ utilizan la funcionalidad de agrupación de conexiones proporcionada por el adaptador de recursos IBM MQ. Consulte Configuración de propiedades del adaptador de recursos de WebSphere MQ para obtener más información.

“Ejemplos de uso de la agrupación de conexiones” en la página 316 explica cómo las aplicaciones que realizan mensajería de salida, y los puertos de escucha, utilizan la agrupación libre al crear las conexiones JMS.

“Hebras de mantenimiento de la agrupación de conexiones libres” en la página 319 explica lo que sucede en estas conexiones cuando una aplicación, o un puerto de escucha, ha terminado con las conexiones.

“Ejemplos de hebras de mantenimiento de agrupaciones” en la página 320 explica cómo se borra la agrupación de conexiones libres para evitar que las conexiones JMS se queden obsoletas.

WebSphere Application Server tiene un límite sobre el número de conexiones que se pueden crear desde una fábrica, especificado por la propiedad *maximum connections* de la fábrica de conexiones. El valor predeterminado de esta propiedad es 10, lo que significa que puede haber hasta 10 conexiones creadas a partir de una fábrica en cualquier momento.

Cada fábrica tiene una agrupación de conexiones libres asociada. Cuando arranca el servidor de aplicaciones, las agrupaciones de conexiones libres están vacías. El número máximo de conexiones que pueden existir en la agrupación libre de una fábrica también se especifica en la propiedad Máximo de conexiones.

Consejo: En JMS 2.0, se puede utilizar una fábrica de conexiones para crear tanto conexiones como contextos. Como resultado, es posible tener una agrupación de conexiones asociada a una fábrica de

conexiones que contenga una mezcla de conexiones y contextos. Se recomienda que una fábrica de conexiones solo se utilice para crear conexiones o crear contextos. Esto garantiza que la agrupación de conexiones de dicha fábrica de conexiones solo contenga objetos de un tipo, lo que hace que la agrupación sea más eficiente.

Para obtener más información sobre cómo funciona la técnica de agrupación de conexiones en WebSphere Application Server, consulte Configuración de la técnica de agrupación de conexiones para conexiones JMS. Para otros servidores de aplicaciones, consulte su correspondiente documentación.

Cómo se utiliza la agrupación de conexiones

Cada fábrica de conexiones JMS tiene una agrupación de conexiones asociada y la agrupación de conexiones contiene cero o más conexiones JMS. Cada conexión JMS tiene una agrupación de sesiones JMS agrupada y cada agrupación de sesiones JMS contiene cero o más sesiones JMS.

Figura 42 en la página 313 muestra la relación entre estos objetos.

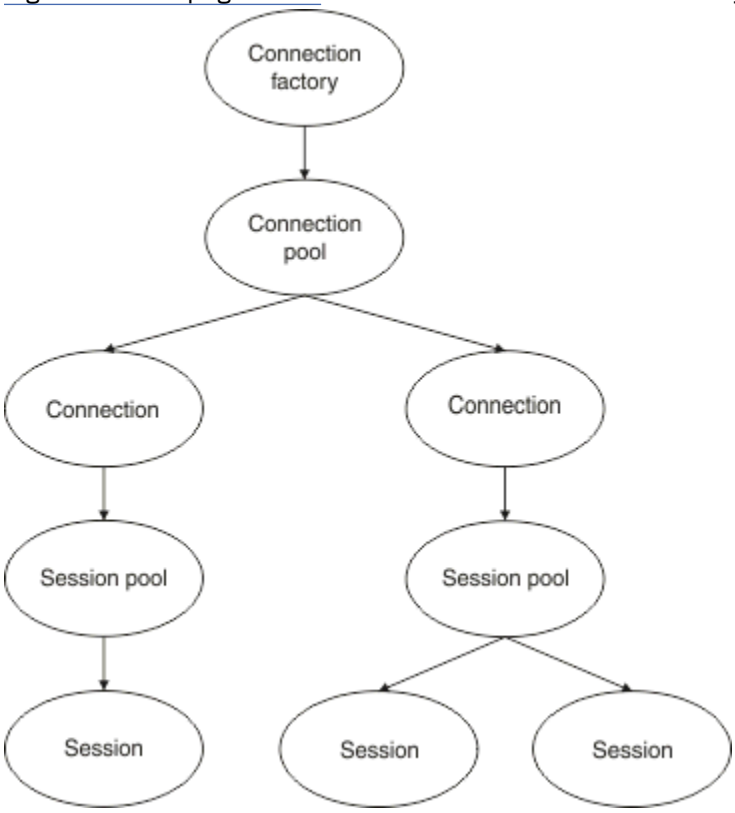


Figura 42. Agrupaciones de conexiones y agrupaciones de sesiones

Cuando se inicia un puerto de escucha, o una aplicación que desea realizar mensajería de salida utiliza la fábrica para crear una conexión, el puerto o la aplicación invoca uno de los métodos siguientes:

- **`connectionFactory.createConnection()`**
- **`ConnectionFactory.createConnection(String, String)`**
- **`QueueConnectionFactory.createQueueConnection()`**
- **`QueueConnectionFactory.createQueueConnection(String, String)`**
- **`TopicConnectionFactory.createTopicConnection()`**
- **`TopicConnectionFactory.createTopicConnection(String, String)`**

El gestor de conexiones WebSphere Application Server intenta obtener una conexión de la agrupación libre para esta fábrica y la devuelve a la aplicación.

Si no hay conexiones libres en la agrupación y el número de conexiones creadas a partir de esta fábrica no ha alcanzado el límite especificado en la propiedad *máximo de conexiones* de dicha fábrica, Connection Manager crea una conexión para que la utilice la aplicación.

Sin embargo, si una aplicación intenta crear una conexión, pero el número de conexiones creadas a partir de esta fábrica ya es igual a la propiedad *máximo de conexiones* de la fábrica, la aplicación esperará a que haya una conexión disponible (que tendrá que devolverse a la agrupación libre).

El tiempo que la aplicación espera se especifica en la propiedad *tiempo de espera de conexión* de la agrupación de conexiones, que tiene un valor predeterminado de 180 segundos. Si se devuelve una conexión a la agrupación libre durante este período de 180 segundos, el gestor de conexiones la volverá a sacar de la agrupación y se la pasará a la aplicación. Sin embargo, si se agota el tiempo de espera, se generará la excepción *ConnectionWaitTimeoutException*.

Cuando una aplicación ha terminado con la conexión y la cierra invocando:

- **`Connection.close()`**
- **`QueueConnection.close()`**
- **`TopicConnection.close()`**

la conexión se mantiene en realidad abierta y se devuelve a la agrupación libre para que otra aplicación pueda reutilizarla. Por lo tanto, puede tener conexiones abierta entre WebSphere Application Server y el proveedor JMS, incluso si no hay aplicaciones JMS ejecutándose en el servidor de aplicaciones.

Propiedades avanzadas de una agrupación de conexiones
Hay una serie de propiedades avanzadas que se pueden utilizar para controlar el comportamiento de las agrupaciones de conexiones de JMS.

Protección contra avalancha

"Cómo usan la agrupación de conexiones las aplicaciones que realizan mensajería de salida" en la página 318 describe el uso del método `sendMessage()`, que incorpora `connectionFactory.createConnection()`.

Suponga una situación en la que hay 50 EJB creando todos ellos conexiones JMS de la misma fábrica de conexiones como parte de su método `ejbCreate()`.

Si todos estos beans se crean a la vez y no hay ninguna conexión en la agrupación de conexiones libres de la fábrica, el servidor de aplicaciones intenta crear 50 conexiones JMS con el mismo proveedor JMS simultáneamente. El resultado es una carga significativa tanto en WebSphere Application Server, como en el proveedor JMS.

Las propiedades de la protección contra avalancha pueden evitar esta situación limitando el número de conexiones JMS que se pueden crear desde una fábrica de conexiones en cualquier momento y escalonando la creación de conexiones adicionales.

La limitación del número de conexiones JMS en cualquier momento se consigue utilizando dos propiedades:

- Umbral de avalancha.
- Intervalo de creación de avalancha.

Cuando una aplicación EJB intenta crear una conexión JMS a partir de una fábrica de conexiones, el gestor de conexiones comprueba cuántas conexiones se están creando. Si ese número es menor o igual que el valor de la propiedad `surge threshold` , el gestor de conexiones continúa abriendo nuevas conexiones.

Sin embargo, si el número de conexiones que se están creando supera la propiedad `surge threshold` , el gestor de conexiones espera durante el periodo de tiempo especificado por la propiedad `surge creation interval` antes de crear y abrir una nueva conexión.

Conexiones atascadas

Una conexión JMS se considera `stuck`, si una aplicación JMS utiliza dicha conexión para enviar una solicitud al proveedor JMS y el proveedor no responde en un periodo de tiempo determinado.

WebSphere Application Server proporciona una forma de detectar conexiones de `stuck` JMS Para utilizar esta función, debe establecer tres propiedades:

- Temporizador de tiempo de atasco.
- Tiempo de atasco.
- Umbral de atasco.

"Ejemplos de hebras de mantenimiento de agrupaciones" en la página 320 explica cómo se ejecuta periódicamente la hebra de mantenimiento de la agrupación y comprueba el contenido de la agrupación libre de una fábrica de conexiones, buscando conexiones que no se hayan utilizado durante un periodo de tiempo, o que hayan existido durante demasiado tiempo.

Para detectar conexiones atascadas, el servidor de aplicaciones también gestiona una hebra de conexión atascada que comprueba el estado de todas las conexiones activas creadas desde una fábrica de conexiones, para ver si alguna de ellas está esperando una respuesta del proveedor JMS.

Cuando la ejecución de la hebra de conexión atascada está determinada por la propiedad `Stuck time timer` . El valor predeterminado de esta propiedad es cero, lo que significa que la detección de conexiones atascadas nunca se ejecuta.

Si la hebra encuentra una en espera de una respuesta, determina cuánto tiempo ha estado esperando y compara esta hora con el valor de la propiedad `Stuck time` .

Si el tiempo que tarda el proveedor JMS en responder supera el tiempo especificado por la propiedad `Stuck time` , el servidor de aplicaciones marca la conexión JMS como atascada.

Por ejemplo, supongamos que la fábrica de conexiones `jms/CF1` tiene la propiedad `Stuck time timer` establecida en 10 y la propiedad `Stuck time` establecida en 15.

La hebra de conexión atascada se activa cada 10 segundos y comprueba si alguna conexión creada desde `jms/CF1` ha estado esperando más de 15 segundos para obtener una respuesta de IBM MQ.

Suponga que un EJB crea una conexión JMS con IBM MQ utilizando `jms/CF1` y, después, intenta crear una sesión JMS utilizando dicha conexión llamando `Connection.createSession()`.

Sin embargo, algo impide que el proveedor JMS responda la solicitud. Quizás la máquina se ha colgado, o un proceso que se ejecuta en el proveedor JMS está bloqueado, lo que impide que se procese cualquier nuevo trabajo.

Diez segundos después de que el EJB invoque `Connection.createSession()`, el temporizador de conexiones atascadas pasa a estar activo y mira las conexiones activas creadas desde `jms/CF1`.

Supongan que solo hay una conexión activa llamada, por ejemplo, `c1`. El primer EJB ha estado esperando 10 segundos una respuesta a una solicitud que ha enviado a `c1`, que es menor que el valor de `Stuck time`, por lo que el temporizador de conexión atascada ignora esta conexión y pasa a estar inactivo.

10 segundos después, la hebra de conexiones atascadaas vuelve a activarse y mira las conexiones activas de `jms/CF1`. Como antes, suponga que solo hay una conexión, `c1`.

Ahora han pasado 20 segundos desde que el primer EJB invocó `createSession()` y sigue esperando una respuesta. 20 segundos es más largo que el tiempo especificado en la propiedad `Stuck time` , por lo que la hebra de conexión atascada marca `c1` como atascada.

Si, cinco segundos después, al final IBM MQ responde y permite que el primer EJB cree una sesión JMS, la conexión se vuelve a utilizar.

El servidor de aplicaciones cuenta el número de conexiones JMS creadas a partir de una fábrica de conexiones que están atascadas. Cuando una aplicación utiliza dicha fábrica de conexiones para crear una nueva JMS Connection, y no hay conexiones libres en la agrupación libre de dicha fábrica, el gestor de conexiones compara el número de conexiones atascadas con el valor de la propiedad `Stuck threshold` .

Si el número de conexiones atascadas es menor que el valor establecido para la propiedad `Stuck threshold`, el gestor de conexiones crea una nueva conexión y la proporciona a la aplicación.

Sin embargo, si el número de conexiones atascadas es igual al valor de la propiedad `Stuck threshold` , la aplicación obtiene una excepción de recurso.

Particiones de agrupación

WebSphere Application Server proporciona dos propiedades que le permiten particionar la agrupación de conexiones libres para una fábrica de conexiones:

- `Number of free pool partitions` indica al servidor de aplicaciones en cuántas particiones desea dividir la agrupación de conexiones libre.
- `Free pool distribution table size` determina cómo se indexan las particiones.

Deje estas propiedades en sus valores predeterminados de cero, a menos que el centro de soporte de IBM le pida que las cambie.

Tenga en cuenta que WebSphere Application Server tiene una propiedad de agrupación de conexiones avanzada adicional denominada `Number of shared partitions`. Esta propiedad especifica el número de particiones que se utilizan para almacenar las conexiones compartidas. Sin embargo, como las conexiones JMS nunca se comparten, esta propiedad no se aplica.

Ejemplos de uso de la agrupación de conexiones
El componente de puerto de escucha de bean controlado por mensaje y las aplicaciones que realizan mensajería de salida utilizan una agrupación de conexiones de JMS.

Figura 43 en la página 316 muestra cómo funciona la agrupación de conexiones en WebSphere Application Server V7.5 y V8.0.

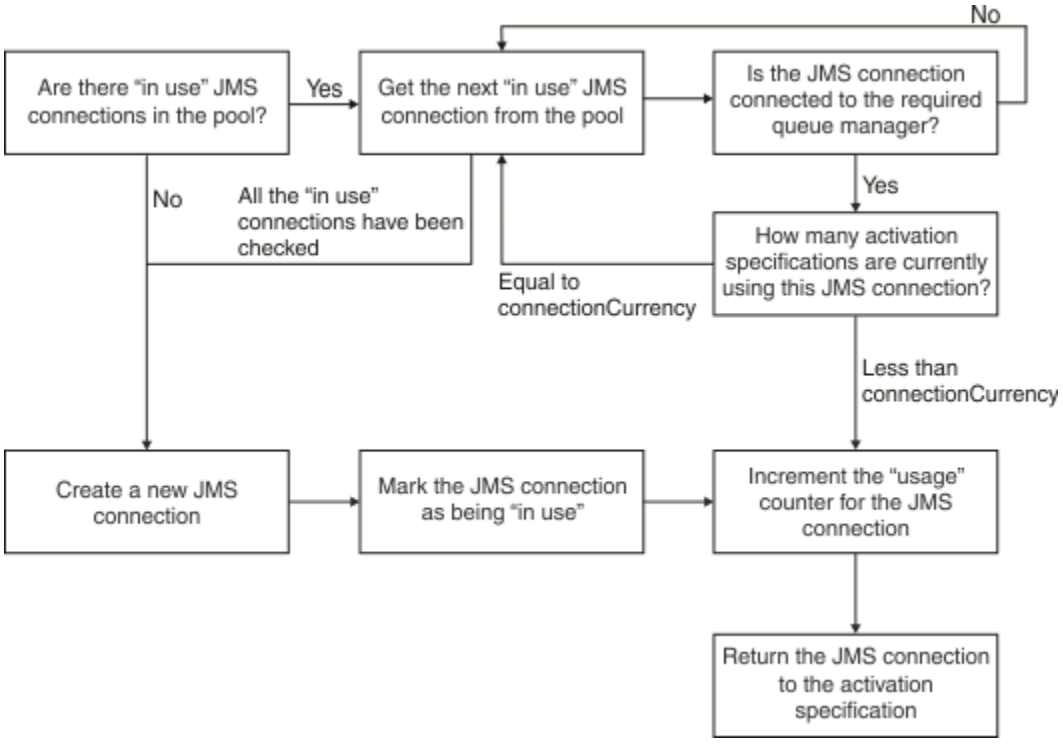


Figura 43. WebSphere Application Server V7.5 y V8.0 - cómo funciona la agrupación de conexiones

Figura 44 en la página 317 muestra cómo funciona la agrupación de conexiones en WebSphere Application Server V8.5.

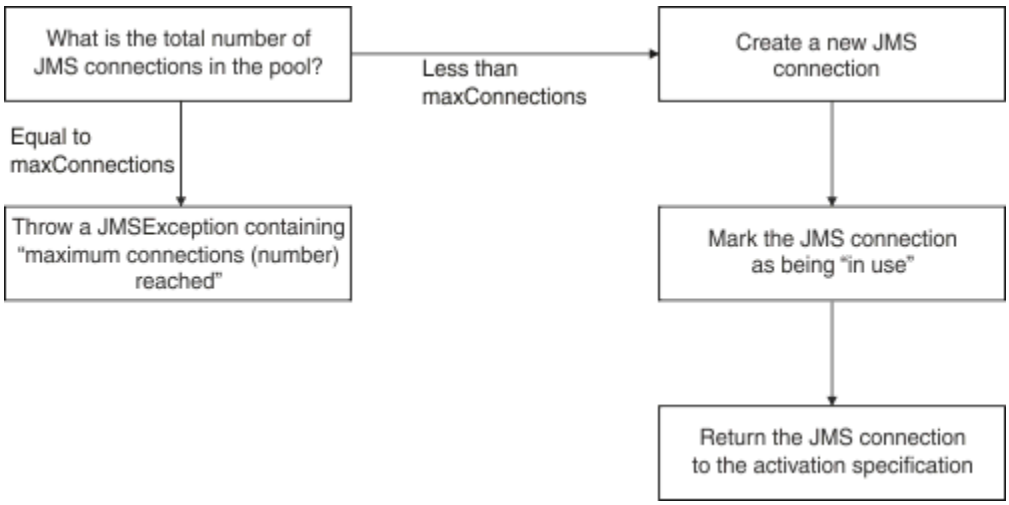


Figura 44. WebSphere Application Server V8.5 - cómo funciona la agrupación de conexiones

Cómo los puertos de escucha MDB usan la agrupación de conexiones

Suponga que tiene un MDB desplegado en un sistema de WebSphere Application Server Network Deployment, que está utilizando IBM MQ como proveedor JMS. El MDB está desplegado contra un puerto de escucha que usa una fábrica de conexiones llamada, por ejemplo, `jms/CF1`, que tiene la propiedad *máximo de conexiones* establecida a dos, lo que significa que solo se pueden crear dos conexiones en esta fábrica en cualquier momento dado.

Cuando se inicia el puerto de escucha, el puerto intenta crear una conexión con IBM MQ, utilizando la fábrica de conexiones `jms/CF1`.

Para ello, el puerto solicita una conexión al gestor de conexiones. Puesto que esta es la primera vez que se utiliza la fábrica de conexiones `jms/CF1`, no hay conexiones en la agrupación de conexiones libres `jms/CF1`, por lo que el gestor de conexiones crea una llamada, por ejemplo, `c1`. Tenga en cuenta que esta conexión existirá durante toda la vida del puerto de escucha.

Ahora, suponga la situación en que detiene el puerto de escucha utilizando la consola de administración de WebSphere Application Server. En este caso, el gestor de conexiones devuelve la conexión a la agrupación libre. Sin embargo, la conexión con IBM MQ permanece abierta.

Si se reinicia el puerto de escucha, este volverá a pedir al gestor de conexiones una conexión con el gestor de colas. Puesto que ahora se tiene una conexión (`c1`) en la agrupación libre, el gestor de conexiones saca esta conexión de la agrupación y la pone a disposición del puerto de escucha.

Ahora suponga que tiene un segundo MDB desplegado en el servidor de aplicaciones y que está utilizando un puerto de escucha distinto.

Suponga que luego intenta iniciar un tercer puerto de escucha, que también está configurado para utilizar la fábrica de conexiones `jms/CF1`. El tercer puerto de escucha solicita una conexión al gestor de conexiones, que busca en la agrupación libre de `jms/CF1` y se encuentra con que está vacía. A continuación, comprueba cuántas conexiones están ya creadas desde la fábrica `jms/CF1`.

Puesto que la propiedad de máximo de conexiones de `jms/CF1` está establecida a 2, y ya se han creado 2 conexiones en esta fábrica, el gestor de conexiones espera 180 segundos (el valor predeterminado de la propiedad de agotamiento del tiempo de espera de una conexión) a que una conexión quede disponible.

Sin embargo, si se para el primer puerto de escucha, su conexión `c1` se coloca en la agrupación libre de `jms/CF1`. El gestor de conexiones recupera esta conexión y se la proporciona al tercer escucha.

Si ahora se intenta reiniciar el primer escucha, este tendrá que esperar a que se pare uno de los otros puertos de escucha para poder reiniciarse. Si no se para ninguno de los puertos de escucha en ejecución en 180 segundos, el primer escucha recibirá un error `ConnectionWaitTimeoutException` y se parará.

Cómo usan la agrupación de conexiones las aplicaciones que realizan mensajería de salida

En esta opción, suponga que hay un único EJB llamado, por ejemplo EJB1, instalado en el servidor de aplicaciones. El bean implementa un método llamado `sendMessage()` que:

- Creación de una conexión de JMS con IBM MQ desde una fábrica `jms/CF1`, utilizando `connectionFactory.createConnection()`.
- Crea una sesión JMS a partir de la conexión.
- Crea un productor de mensajes a partir de la sesión.
- Envía un mensaje.
- Cierra el productor.
- Cierra la sesión.
- Cierra la conexión, invocando `connection.close()`.

Suponga que la agrupación libre de la fábrica `jms/CF1` está vacía. Cuando se invoca el EJB por primera vez, el bean intenta crear una conexión con IBM MQ desde la fábrica `jms/CF1`. Puesto que la agrupación libre de la fábrica está vacía, el gestor de conexiones crea una conexión y se la proporciona a EJB1.

Justo antes de salir el método, este invoca `connection.close()`. En lugar de cerrar `c1`, el gestor de conexiones toma la conexión y la coloca en la agrupación libre de `jms/CF1`.

La próxima vez que se llame a `sendMessage()` , el método `connectionFactory.createConnection()` devuelve `c1` a la aplicación.

Suponga que tiene una segunda instancia del EJB que ejecuta al mismo tiempo que la primera. Cuando ambas instancias están invocando `sendMessage()`, se crean dos conexiones a partir de la fábrica de conexiones `jms/CF1`.

Ahora suponga que se crea una tercera instancia del bean. Cuando el tercer bean invoque `sendMessage()`, el método llama a `connectionFactory.createConnection()` para crear una conexión desde `jms/CF1`.

Sin embargo, en este momento hay dos conexiones creadas a partir de `jms/CF1`, que es igual al valor del máximo de conexiones de esta fábrica. Por tanto, el método `createConnection()` espera durante 180 segundos (el valor predeterminado de la propiedad de tiempo de espera de conexión) a que una conexión esté disponible.

Sin embargo, si el método `sendMessage()` del primer EJB llama a `connection.close()` y sale, la conexión que estaba utilizando, `c1`, se devuelve a la agrupación de conexiones libres. El gestor de conexiones vuelve a sacar la conexión de la agrupación libre y la entrega al tercer EJB. Después, la llamada de ese bean a `connectionFactory.createConnection()` retorna, lo que permite que el método `sendMessage()` complete.

Puertos de escucha de MDB y EJBs que utilizan la misma agrupación de conexiones

Los dos ejemplos anteriores muestran cómo los puertos de escucha y los EJB pueden utilizar la agrupación de conexiones de forma aislada. Sin embargo, puede tener tanto el puerto de escucha y un EJB ejecutándose en el mismo servidor de aplicaciones y creando conexiones JMS utilizando la misma fábrica de conexiones.

Hay que tener en cuenta las implicaciones de esta situación

Lo importante es recordar que la fábrica de conexiones está compartida entre el puerto de escucha y el EJB.

Por ejemplo, suponga que tiene un escucha y un EJB que ejecutan al mismo tiempo. Ambos están utilizando la fábrica de conexiones `jms/CF1`, lo que significa que se ha alcanzado el límite de conexiones especificado en la propiedad de número máximo de conexiones de dicha fábrica.

Si intenta iniciar otro puerto de escucha, u otra instancia de un EJB, tendrá que esperar a que se devuelva una conexión a la agrupación de conexiones libres de `jms/CF1`.

Hebras de mantenimiento de la agrupación de conexiones libres
Asociada a cada agrupación de conexiones libres hay una hebra de mantenimiento de agrupaciones, que supervisa la agrupación libre para garantizar que las conexiones en ella sigan siendo válidas.

Si la hebra de mantenimiento de la agrupaciones decide que hay que descartar una conexión de la agrupación libre, cierra físicamente la conexión JMS con IBM MQ.

Cómo funciona la hebra de mantenimiento de agrupaciones

El comportamiento de la hebra de mantenimiento de agrupaciones viene determinado por el valor de cuatro propiedades de una agrupación de conexiones:

Tiempo de espera de caducidad
Tiempo que una conexión permanece abierta.

Mínimo de conexiones
Número mínimo de conexiones que el gestor de conexiones mantiene en la agrupación libre de una fábrica de conexiones.

Tiempo de recogida
Frecuencia con la que ejecuta la hebra de mantenimiento.

Tiempo de espera de inactividad
Tiempo que una conexión permanece en la agrupación libre antes de ser cerrada.

De forma predeterminada, la hebra mantenida de la agrupación se ejecuta cada 180 segundos, aunque este valor se puede cambiar estableciendo la propiedad **Reap time** de la agrupación de conexiones.

La hebra de mantenimiento examina cada una de las conexiones de la agrupación, comprueba durante cuánto tiempo ha estado en la agrupación y cuánto tiempo ha transcurrido desde que se creó y utilizó por última vez.

Si la conexión no se ha utilizado durante un periodo más largo que el valor de la propiedad **Unused timeout** para la agrupación de conexiones, la hebra de mantenimiento comprueba el número de conexiones que hay actualmente en la agrupación libre. Si dicho número es:

- Mayor que el valor de **Minimum connections**, el gestor de conexiones cierra la conexión.
- Es igual al valor de **Minimum connections**, la conexión no está cerrada y permanece en la agrupación libre.

El valor predeterminado de la propiedad **Minimum connections** es *1*, lo que significa que, por razones de rendimiento, el gestor de conexiones siempre intenta mantener al menos una conexión en la agrupación libre.

La propiedad **Unused timeout** tiene un valor predeterminado de 1800 segundos. De forma predeterminada, si una conexión se devuelve a la agrupación libre y no se vuelve a utilizar durante al menos 1800 segundos, dicha conexión se cierra, siempre que al cerrarse quede al menos una conexión en la agrupación libre.

Este procedimiento evita que las conexiones no utilizadas se vuelvan obsoletas. Para desactivar esta característica, establezca la propiedad **Unused timeout** en cero.

Si una conexión está en la agrupación libre y el tiempo transcurrido desde su creación es mayor que el valor de la propiedad **Aged timeout** para la agrupación de conexiones, se cierra independientemente del tiempo que haya pasado desde que se utilizó por última vez.

De forma predeterminada, la propiedad **Aged timeout** se establece en cero, lo que significa que la hebra de mantenimiento nunca realiza esta comprobación. Las conexiones que han existido durante más tiempo que la propiedad **Aged timeout** se descartan independientemente de cuántas conexiones permanezcan en la agrupación libre. Tenga en cuenta que la propiedad **Minimum connections** no tiene ningún efecto en esta situación.

Inhabilitación de la hebra de mantenimiento de la agrupaciones

De la descripción anterior se deduce que la hebra de mantenimiento de agrupaciones trabaja mucho cuando está activa, en particular cuando hay un gran número de conexiones en la agrupación libre de la fábrica de conexiones.

Por ejemplo, supongamos que hay tres fábricas de conexiones JMS , con la propiedad **Maximum connections** establecida en 10 para cada fábrica. Cada 180 segundos, tres hebras de mantenimiento de agrupaciones se activan y exploran cada una su correspondiente agrupación libre de cada fábrica de conexiones. Si las agrupaciones libres tienen muchas conexiones, las hebras de mantenimiento tendrán mucho trabajo que hacer, lo que puede penalizar significativamente el rendimiento.

Puede inhabilitar la hebra de mantenimiento de agrupación para una agrupación de conexiones libre individual estableciendo su propiedad **Reap time** en cero.

La inhabilitación de la hebra de mantenimiento significa que las conexiones nunca se cierran, incluso si ha transcurrido el **Unused timeout** . Sin embargo, las conexiones se pueden seguir cerrando si el **Aged timeout** ha pasado.

Cuando una aplicación ha finalizado con una conexión, el gestor de conexiones comprueba cuánto tiempo ha existido la conexión y, si ese periodo es más largo que el valor de la propiedad **Aged timeout** , el gestor de conexiones cierra la conexión en lugar de devolverla a la agrupación libre.

Implicaciones transaccionales del tiempo de espera de caducidad

Tal como se describe en la sección anterior, la propiedad **Aged timeout** especifica cuánto tiempo permanece abierta una conexión con el proveedor de JMS antes de que el gestor de conexiones la cierre.

El valor predeterminado para la propiedad **Aged timeout** es cero, lo que significa que la conexión nunca se cerrará porque es demasiado antigua. Debe dejar la propiedad **Aged timeout** en este valor, ya que la habilitación de **Aged timeout** puede tener implicaciones transaccionales al utilizar JMS dentro de los EJB.

En JMS, la unidad de una transacción es una JMS sesión de , que se crea a partir de una *conexión*de JMS . Es la JMS sesión de que se incluye en las transacciones, y no la JMS conexión de .

Debido al diseño del servidor de aplicaciones, las conexiones de JMS se pueden cerrar porque ha transcurrido el **Aged timeout** , incluso si las sesiones de JMS creadas a partir de esa conexión están implicadas en una transacción.

El cierre de una conexión JMS provoca que se retrotraiga cualquier trabajo transaccional pendiente en las sesiones JMS, tal como se describe en la especificación JMS. Sin embargo, el servidor de aplicaciones no es consciente de que las sesiones JMS creadas a partir de la conexión ya no son válidas. Cuando el servidor intenta utilizar la sesión para confirmar o retrotraer una transacción, se produce una excepción `IllegalStateException`.

Importante: Si desea utilizar **Aged timeout** con conexiones JMS desde dentro de EJB, asegúrese de que cualquier trabajo JMS se confirme explícitamente en la sesión JMS , antes de que el método EJB que realiza las salidas de operaciones JMS .

Ejemplos de hebras de mantenimiento de agrupaciones
Utilizando el ejemplo EJB (Enterprise JavaBean) para comprender cómo funciona la hebra de mantenimiento de la agrupación. Tenga en cuenta que también puede utilizar beans controlados por mensajes (MDB) y puertos de escucha, ya que todo lo que necesita es una forma de obtener conexiones en la agrupación libre.

Consulte “Cómo usan la agrupación de conexiones las aplicaciones que realizan mensajería de salida” en la página 318 si desea más detalles del método `sendMessage()`.

Ha configurado la fábrica de conexiones con los siguientes valores:

- **Reap time** en su valor predeterminado de 180 segundos
- **Aged timeout** en su valor predeterminado de cero segundos

- **Unused timeout** establecido en 300 segundos

Una vez arrancado el servidor de aplicaciones, se invocará el método `sendMessage()`.

Este método crea una conexión llamada, por ejemplo, `c1`, usando la fábrica `jms/CF1`, usa dicha fábrica para enviar un mensaje y luego llama `connection.close()`, que provoca que `c1` se coloque en la agrupación libre.

Tras 180 segundos, la hebra de mantenimiento de agrupacionesn se inicia y mira en la agrupación de conexiones libres `jms/CF1`. Hay una conexión libre `c1` en la agrupación, por lo que la hebra de mantenimiento mira el momento en que se ha devuelto la conexión y lo compara con la hora actual.

Han pasado 180 segundos desde que se puso la conexión en la agrupación libre, que es menor que el valor de la propiedad **Unused timeout** para `jms/CF1`. Por tanto, la hebra de mantenimiento no hace nada con la conexión.

180 segundos más tarde, la hebra de mantenimiento de agrupaciones se ejecuta de nuevo. La hebra de mantenimiento busca la conexión `c1`y determina que la conexión ha estado en la agrupación durante 360 segundos, que es más larga que el valor **Unused timeout** establecido, por lo que el gestor de conexiones cierra la conexión.

Si ahora vuelve a ejecutar el método `sendMessage()`, cuando la aplicación invoque a `connectionFactory.createConnection()`, el gestor de conexiones crea una nueva conexión con IBM MQ porque la agrupación de conexiones libres para la fábrica de conexiones está vacía.

El ejemplo anterior muestra cómo la hebra de mantenimiento utiliza las propiedades **Reap time** y **Unused timeout** para evitar conexiones obsoletas, cuando la propiedad **Aged timeout** se establece en cero.

¿Cómo funciona la propiedad **Aged timeout** ?

En el ejemplo siguiente, suponga que ha establecido:

- Propiedad **Aged timeout** a 300 segundos
- Propiedad **Unused timeout** a cero.

Invoca el método `sendMessage()` y este intenta crear una conexión a partir de la fábrica de conexiones `jms/CF1`.

Puesto que la agrupación libre de esta fábrica está vacía, el gestor de conexiones crea una nueva conexión, `c1` y la devuelve a la aplicación. Cuando `sendMessage()` invoca `connection.close()`, `c1` se devuelve a la agrupación de conexiones libres.

180 segundos más tarde, se ejecuta la hebra de mantenimiento de agrupaciones. La hebra encuentra `c1` en la agrupación de conexiones libres y comprueba cuánto hace que se ha creado. La conexión ha existido durante 180 segundos, que es menor que **Aged timeout**, por lo que la hebra de mantenimiento de la agrupación la deja sola y vuelve a estar en suspensión.

60 segundos más tarde, `sendMessage()` se invoca de nuevo. Esta vez, cuando el método llama a `connectionFactory.createConnection()`, el gestor de conexiones descubre que hay una conexión, `c1`, disponible en la agrupación libre para `jms/CF1`. El gestor de conexiones saca `c1` de la agrupación libre y se la da a la aplicación.

La conexión se devuelve a la agrupación libre cuando `sendMessage()` devuelve el control. 120 segundos más tarde, la hebra de mantenimiento de agrupaciones se despierta de nuevo, explora el contenido de la agrupación libre de `jms/CF1` y descubre `c1`.

Aunque la conexión sólo se ha utilizado hace 120 segundos, la hebra de mantenimiento de la agrupación cierra la conexión, porque la conexión ha existido durante un total de 360 segundos, que es superior al valor de 300 segundos que ha establecido para la propiedad **Aged timeout** .

Cómo afecta la propiedad Mínimo de conexiones a la hebra de mantenimiento de agrupacioens

Volviendo a utilizar el ejemplo de "Cómo los puertos de escucha MDB usan la agrupación de conexiones" en la página 317, suponga que tiene dos MDB desplegados en el servidor de aplicaciones, cada uno de los cuales está utilizando un puerto de escucha diferente.

Cada puerto de escucha está configurado para utilizar la fábrica de conexiones `jms/CF1`, que se ha configurado con:

- Propiedad **Unused timeout** establecida en 120 segundos
- Propiedad **Reap time** establecida en 180 segundos
- Propiedad **Minimum connections** establecida en 1

Suponga que el primer escucha se para y su conexión `c1` se coloca en la agrupación libre. 180 segundos después, la hebra de mantenimiento de agrupación se activa, explora el contenido de la agrupación libre para `jms/CF1`y descubre que `c1` ha estado en la agrupación libre durante más tiempo que el valor de la propiedad **Unused timeout** para la fábrica de conexiones.

Sin embargo, antes de cerrar `c1`, la hebra de mantenimiento mira cuántas conexiones quedarán en la agrupación si se descarta esta conexión. Puesto que `c1` es la única conexión de la agrupación de conexiones libre, el gestor de conexiones no la cierra, ya que hacerlo haría que el número de conexiones que permanecen en la agrupación libre fuera menor que el valor establecido para **Minimum connections**.

Ahora suponga que el segundo escucha está parado. La agrupación de conexiones libres contiene ahora dos conexiones, `c1` y `c2`.

180 segundos más tarde, la hebra de mantenimiento de agrupaciones se ejecuta de nuevo. En ese momento, `c1` ha estado en la agrupación de conexiones libres durante 360 segundos y `c2` durante 180 segundos.

La hebra de mantenimiento de agrupación comprueba `c1` y descubre que ha estado en la agrupación durante más tiempo que el valor de la propiedad **Unused timeout** .

A continuación, la hebra comprueba cuántas conexiones hay en la agrupación libre y las compara con el valor de la propiedad **Minimum connections** . Puesto que la agrupación contiene dos conexiones y **Minimum connections** se establece en 1, el gestor de conexiones cierra `c1`.

La hebra de mantenimiento mira ahora `c2`. También ha estado en la agrupación de conexiones libre durante más tiempo que el valor de la propiedad **Unused timeout** . Sin embargo, dado que el cierre de `c2` dejaría la agrupación de conexiones libres con menos conexiones de las configuradas en el mínimo de conexiones, el gestor de conexiones no hace nada con `c2`.

Conexiones de JMS y IBM MQ
Información sobre el uso de IBM MQ como proveedor de JMS.

Uso del transporte de enlaces

Si se ha configurado una fábrica de conexiones para que utilice el transporte de enlaces, cada conexión de JMS establece una conversación (también conocida como **hconn**) con IBM MQ. La conversación utiliza la comunicación entre procesos (o la memoria compartida) para comunicarse con el gestor de colas.

Utilización del transporte de cliente

Cuando se ha configurado una fábrica de conexiones del proveedor de mensajería de IBM MQ para que utilice el transporte de cliente, cada conexión creada desde dicha fábrica establecerá una nueva conexión (también conocida como **hconn**) aIBM MQ.

Para las fábricas de conexiones que se conectan a un gestor de colas utilizando la modalidad normal del proveedor de mensajería de IBM MQ, es posible que se creen varias conexiones de JMS desde la fábrica

de conexiones para compartir una conexión TCP/IP con IBM MQ. Para obtener más información, consulte el apartado “Compartir una conexión TCP/IP en IBM MQ classes for JMS” en la página 325.

Para determinar el número máximo de canales cliente utilizados por las conexiones JMS en un momento dado, sume el valor de la propiedad *Conexiones máximas* de todas las fábricas de conexiones que apuntan al mismo gestor de cola.

Por ejemplo, suponga que tiene dos fábricas de conexiones, `jms/CF1` y `jms/CF2`, que se han configurado para la conexión al mismo gestor de cola de IBM MQ usando el mismo canal IBM MQ.

Estas fábricas están utilizando las propiedades de agrupación de conexiones predeterminadas, lo que significa que *Conexiones máximas* se establece en 10. Si todas las conexiones se están utilizando desde jms/CF1 y jms/CF2 al mismo tiempo, habrá 20 conversaciones entre el servidor de aplicaciones y IBM MQ.

Si la fábrica de conexiones se conecta al gestor de colas utilizando la modalidad normal del proveedor de mensajería de IBM MQ, el número máximo de conexiones TCP/IP que pueden haber entre el servidor de aplicaciones y el gestor de colas para estas fábricas de conexiones es:

```
20/the value of SHARECNV for the IBM MQ channel
```

Si la fábrica de conexiones se ha configurado para conectarse utilizando la modalidad de migración del proveedor de mensajería de IBM MQ, el número máximo de conexiones TCP/IP entre el servidor de aplicaciones y IBM MQ para estas fábricas de conexiones sería 20 (uno para cada conexión JMS en las agrupaciones de conexiones para las dos fábricas).

Conceptos relacionados

“Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85
IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

Agrupación de objetos en un entorno de Java SE

Con Java SE (o con otra infraestructura, como Spring) los modelos de programación son extremadamente flexibles. Por tanto, una única estrategia de agrupamientos no valdría para todos. Debe tener en cuenta si existe una infraestructura que pueda hacer algún tipo de agrupación, por ejemplo, Spring.

De lo contrario, la lógica de aplicación podría hacerse cargo de ello. Plantéese qué grado de complejidad tiene la aplicación. Es mejor entender la aplicación y lo que exige a la conectividad con el sistema de mensajería. Con frecuencia, las aplicaciones se desarrollan con su propio código envolvente alrededor del API básica de JMS.

Aunque esto puede ser un enfoque muy razonable que puede ocultar la complejidad, vale la pena tener en cuenta que puede ocasionar problemas. Por ejemplo, un método genérico `getMessage()` que se invoque con frecuencia, no debería limitarse a abrir y cerrar consumidores.

Puntos a tener en cuenta:

- ¿Durante cuánto tiempo tiene que acceder la aplicación a IBM MQ? Todo el tiempo, o solo ocasionalmente.
- ¿Con qué frecuencia se van a enviar los mensajes? Cuanto menor sea la frecuencia, más se podrá compartir una única conexión con IBM MQ.
- Una excepción de conexión interrumpida suele ser síntoma de la necesidad de volver a crear una conexión agrupada. Más puntos:
  - Excepciones de seguridad o host no disponible.
  - Excepciones de cola llena.
- Si se produce una excepción de conexión interrumpida, ¿qué debería pasar con las otras conexiones libres de la agrupación? ¿Deberían cerrarse y volver a crearse?

- Si se está usando TLS, por ejemplo, ¿cuánto tiempo se necesita que una única conexión permanezca abierta?
- ¿Cómo se identificará una conexión agrupada a sí misma de forma que un administrador del gestor de colas pueda detectarla y seguirla?

Debe tener en cuenta todos los objetos JMS para la agrupación, y agrupar dichos objetos siempre que sea posible. Entre estos objetos se incluyen:

- Conexiones JMS.
- Session
- Contextos.
- Productores y consumidores de todos los tipos.

Cuando se utiliza el transporte de cliente, las conexiones, sesiones y contextos de JMS utilizarán sockets cuando se comuniquen con el gestor de colas IBM MQ. Al agrupar estos objetos, se reduce el número de conexiones entrantes de IBM MQ (`hConns`) con el gestor de colas y se obtiene una reducción en el número de instancias de canal.

Al utilizar el transporte de enlaces con el gestor de colas, se elimina completamente la capa de red. Sin embargo, muchas aplicaciones utilizan el transporte de cliente para proporcionar una configuración con mayor disponibilidad y con una carga de trabajo más equilibrada.

Los productores y consumidores JMS abren destinos en el gestor de colas. Si se abre un número menor de colas o temas y varias partes de la aplicación están utilizando estos objetos, la agrupación de estos puede ser útil.

Desde una perspectiva de IBM MQ, este proceso guarda una secuencia de operaciones MQOPEN y MQCLOSE.

Conexiones, sesiones y contextos

Todos estos objetos encapsulan los descriptores de conexiones IBM MQ en el gestor de colas y se generan a partir de `ConnectionFactory`. Se puede añadir lógica a una aplicación para limitar a un número determinado las conexiones y otros objetos creados a partir de una única fábrica de conexiones.

Se puede utilizar una estructura de datos simple en la aplicación para que contenga las conexiones creadas. El código de aplicación que necesite usar una de dichas estructuras podrá *sacar* un objeto para usarlo.

Tenga en cuenta los siguientes factores:

- ¿Cuándo hay que eliminar las conexiones de la agrupación? En general, cree un escucha de excepciones sobre la conexión. Cuando se invoque ese escucha para procesar una excepción, se deberá volver a crear la conexión y todas las sesiones creadas a partir ella.
- Si ses usa una CCDT para el equilibrado de cargas de trabajo, las conexiones podrían ir a gestores de colas diferentes. Esto podría aplicarse a los requisitos de la agrupación.

Recuerde que la especificación JMS indica que es un error de programación para varias hebras que van a acceder a una sesión o un contexto a la vez. El código IBM MQ JMS intenta ser riguroso en su manejo de las hebras. Sin embargo, hay que añadir lógica a la aplicación, para asegurarse de que solo una hebra use un objeto de sesión o de contexto a la vez.

Productores y consumidores

Cada productor y consumidor que se crea abre un destino en el gestor de colas. Si se va a utilizar el mismo destino en diversas tareas, tiene sentido mantener abiertos los objetos consumidores o productores. Cierre el objeto únicamente cuando se haya realizado todo el trabajo.

Aunque la apertura y el cierre de un destino son operaciones breves, si se realizan con frecuencia el tiempo empleado se acumula.

El ámbito de estos objetos se circunscribe a la sesión o al contexto en el que se crean, por lo tanto, se tienen que mantener en dicho ámbito. Por lo general, las aplicaciones se escriben de ta forma tal que esto sea fácil de hacer.

Supervisión

¿Cómo supervisarán las aplicaciones sus agrupaciones de objetos? La respuesta está determinada en gran medida por la complejidad de la solución de agrupamiento aplicada.

Si se considera una implementación de agrupación de JavaEE, hay un gran número de opciones, incluyendo:

- El tamaño actual de las agrupaciones.
- El tiempo que los objetos han pasado en ellas.
- La limpieza de las agrupaciones.
- La renovación de las conexiones.

También debe tenerse en cuenta la forma en que aparece una única sesión reutilizada en el gestor de colas. Hay propiedades de fábrica de conexiones para identificar la aplicación (por ejemplo, `appName`) que podrían ser útiles.

"Utilización de IBM MQ classes for JMS/Jakarta Messaging" en la página 85
IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

Compartir una conexión TCP/IP en IBM MQ classes for JMS
Se pueden crear varias instancias de un canal MQI para que compartan una sola conexión TCP/IP.

Las aplicaciones que se ejecutan en el mismo entorno de ejecución de Java y que utilizan el adaptador de recursos IBM MQ classes for JMS o IBM MQ para conectarse a un gestor de colas utilizando el transporte CLIENT, se pueden realizar para compartir una instancia de canal.

Si un canal se define con el parámetro **SHARECNV** establecido en un valor mayor que 1, ese número de conversaciones pueden compartir una instancia de canal. Para habilitar una fábrica de conexiones o una especificación de activación para utilizar esta función, establezca la propiedad **SHARECONVALLOWED** en YES.

Cada conexión JMS y cada sesión JMS que crea una aplicación JMS crea su propia conversación con el gestor de colas.

Cuando se inicia una especificación de activación, el adaptador de recursos de IBM MQ inicia una conversación con el gestor de colas para que la utilice la especificación de activación. Cada sesión de servidor de la agrupación de sesiones de servidor que está asociada a la especificación de activación también inicia una conversación con el gestor de colas.

El atributo **SHARECNV** es un método de mejor esfuerzo para la compartición de conexiones. Por lo tanto, cuando se utiliza un valor de **SHARECNV** mayor que 0 con IBM MQ classes for JMS, no se garantiza que una nueva solicitud de conexión comparta siempre una conexión ya establecida.

Cómo se comparten las conexiones TCP/IP

Hay dos estrategias disponibles para compartir conexiones TCP/IP:

La estrategia GLOBAL
Esta estrategia es la estrategia predeterminada para compartir conexiones TCP/IP. Cualquier conexión o sesión de JMS puede utilizar una conversación en cualquier conexión TCP/IP adecuada. La idoneidad viene determinada por factores como la dirección de host, el número de puerto, el ID de usuario y la contraseña y los parámetros TLS/SSL.

Este enfoque para compartir conexiones TCP/IP minimiza el número de instancias de canal que están en uso, pero a costa de la contención por el acceso a una agrupación global de conexiones TCP/IP.

La estrategia CONNECTION

Con esta estrategia, las instancias de canal sólo se comparten entre objetos JMS relacionados. Específicamente, cuando se crea una conexión de JMS , se crea una instancia de canal para ella, y las conversaciones adicionales en dicha instancia de canal solo están disponibles para las sesiones de JMS creadas por dicha conexión de JMS .

Si se crean más conversaciones de las que especifica el atributo SHARECNV, se crea una nueva instancia de canal que sólo pueden utilizar las sesiones JMS creadas por la conexión JMS original.

Este enfoque para compartir instancias de canal reduce la contienda por las conversaciones, a expensas de que potencialmente se necesiten más instancias de canal.

Especificación explícita de una estrategia de compartición de instancias de canal

V 9.3.2

De forma predeterminada, se utiliza la estrategia GLOBAL si las aplicaciones no se pueden volver a conectar. Las aplicaciones reconectables siempre utilizan la estrategia CONNECTION.

Para las aplicaciones que utilizan IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, la estrategia CONNECTION se puede habilitar para aplicaciones no reconectables en toda la aplicación. Puede habilitar la estrategia CONNECTION estableciendo la propiedad del sistema `com.ibm.mq.jms.channel.sharing` en el valor `CONNECTION`. Este valor no distingue entre mayúsculas y minúsculas y se ignora cualquier valor que no sea `CONNECTION` .

Puede establecer la propiedad del sistema `com.ibm.mq.jms.channel.sharing` de una de las maneras siguientes:

- Establezca la propiedad como parte de la inicialización de JVM utilizando la opción de línea de mandatos "-D":

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- Establezca la propiedad antes de cualquier uso de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging utilizando `System.setProperty()`

Cálculo del número de instancias de canal para la estrategia de compartición GLOBAL

Utilice las fórmulas siguientes para determinar el número máximo de instancias de canal creadas por una aplicación:

Especificaciones de activación

Número de instancias de canal = (*valor_maxPoolDepth* + 1) / *valor_SHARECNV*

Donde *valor_maxPoolDepth* es el valor de la propiedad **maxPoolDepth** y *valor_SHARECNV* es el valor de la propiedad **SHARECNV** en el canal que utiliza la especificación de activación.

Otras aplicaciones de JMS

Número de instancias de canal = (*conexiones_jms* + *sesiones_jms*) / *valor_SHARECNV*

Donde *jms_connections* es el número de conexiones creadas por la aplicación, *jms_sessions* es el número de JMS sesiones creadas por la aplicación y *SHARECNV_value* es el valor de la propiedad **SHARECNV** en el canal que utiliza la especificación de activación.

Cálculo del número de instancias de canal para la estrategia de compartición CONNECTION

El número de instancias de canal depende de la distribución de sesiones de JMS entre las conexiones de JMS en la aplicación.

Permita una conversación para la conexión JMS y una conversación para cada sesión JMS bajo esa conexión JMS y, a continuación, divida por el valor **SHARECNV** , redondeando hacia arriba. Este cálculo proporciona las instancias de canal que necesita esa conexión JMS .

El mismo principio puede aplicarse a las especificaciones de activación. Considere la especificación de activación como una conexión JMS y la propiedad **maxPoolDepth** como el número de sesiones JMS .

Ejemplos

En los ejemplos siguientes, se muestra cómo utilizar las fórmulas para calcular el número de instancias de canal creadas en un gestor de colas por las aplicaciones utilizando IBM MQ classes for JMS o el adaptador de recursos de IBM MQ.

Ejemplo de aplicación JMS
Una conexión de aplicación JMS se conecta a un gestor de colas utilizando el transporte CLIENT y crea una conexión JMS y tres sesiones JMS. El canal que utiliza la aplicación para conectarse al gestor de colas tiene la propiedad **SHARECNV** establecida en un valor 10. Cuando se ejecuta la aplicación, hay cuatro conversaciones entre la aplicación y el gestor de colas y una instancia de canal. Las cuatro conversaciones comparten la instancia de canal.

Ejemplo de especificación de activación
Una especificación de activación se conecta a un gestor de colas utilizando el transporte CLIENT. La especificación de activación se configura con la propiedad **maxPoolDepth** establecida en 10. El canal que se ha configurado en la especificación de activación tiene la propiedad **SHARECNV** establecida en 10. Cuando se ejecuta la especificación de activación y procesa 10 mensajes simultáneamente, el número de conversaciones entre la especificación de activación y el gestor de colas es 11 (10 conversaciones para las sesiones de servidor y una para la especificación de activación). El número de instancias de canal que utiliza la especificación de activación es 2.

Ejemplo de especificación de activación
Una especificación de activación se conecta a un gestor de colas utilizando el transporte CLIENT. La especificación de activación se configura con la propiedad **maxPoolDepth** establecida en 5. El canal que la especificación de activación está configurada para utilizar tiene la propiedad **SHARECNV** establecida en 0. Cuando la especificación de activación se está ejecutando y procesando 5 mensajes simultáneamente, el número de conversaciones entre la especificación de activación y el gestor de colas es 6 (cinco conversaciones para las sesiones de servidor y una para la especificación de activación). El número de instancias de canal que utiliza la especificación de activación es 6 porque la propiedad **SHARECNV** del canal se establece en 0, cada conversación utiliza su propia instancia de canal.

Tareas relacionadas

“Determinación del número de conexiones TCP/IP que se crean de WebSphere Application Server a IBM MQ” en la página 518
Al utilizar la funcionalidad de compartición de conversaciones, varias conversaciones pueden compartir instancias de canal MQI, que también se conoce como una conexión TCP/IP.

Especificación de un rango de puertos para las conexiones de cliente en IBM MQ classes for JMS
Utilice la propiedad LOCALADDRESS para especificar un rango de puertos a los que se puede enlazar aplicación.

Cuando una aplicación de IBM MQ classes for JMS intenta conectar con un gestor de colas de IBM MQ en la modalidad de cliente, un cortafuegos podría permitir solo las conexiones que se originan en un puerto o rango de puertos especificado. En esta situación, puede utilizar la propiedad LOCALADDRESS de un objeto ConnectionFactory, QueueConnectionFactory o bien TopicConnectionFactory para especificar un puerto o un rango de puertos a los que se puede enlazar la aplicación.

Puede establecer la propiedad LOCALADDRESS utilizando la herramienta de administración de IBM MQ JMS, o llamando al método setLocalAddress() en una aplicación JMS. El ejemplo siguiente establece la propiedad desde dentro de una aplicación:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Cuando la aplicación se conecta posteriormente a un gestor de colas, la aplicación se enlaza an una dirección IP local y un número de puerto dentro del rango de 192.0.2.0(2000) a 192.0.2.0(3000).

En un sistema con más de una interfaz de red, también puede utilizar la propiedad LOCALADDRESS para especificar la interfaz de red que se debe utilizar para una conexión.

Para una conexión en tiempo real con un intermediario, la propiedad LOCALADDRESS sólo es aplicable cuando se utiliza la multidifusión. En este caso, puede utilizar la propiedad para especificar la interfaz de red local que se debe utilizar para una conexión, pero el valor de la propiedad no debe contener un número de puerto ni un rango de números de puerto.

Pueden producirse errores de conexión si restringe el rango de puertos. Si se produce un error, se genera una JMSException con una MQException incorporada que contiene el código de razón de IBM MQ MQRC_Q_MGR_NOT_AVAILABLE y el mensaje siguiente:

```
Se ha rechazado el intento de conexión de
socket debido a restricciones de LOCAL_ADDRESS_PROPERTY
```

Puede producirse un error si se están utilizando todos los puertos del rango especificado o si la dirección IP, el nombre de host o el número de puerto especificados no son válidos (un número de puerto negativo, por ejemplo).

Debido a que IBM MQ classes for JMS puede crear conexiones que no sean necesarias para una aplicación, considere siempre la posibilidad de especificar un rango de puertos. En general, todas las sesiones creadas por una aplicación necesitan un puerto y IBM MQ classes for JMS puede necesitar tres o cuatro puertos adicionales. Si se produce un error de conexión, aumente el rango de puertos.

La agrupación de conexiones, que se utiliza de forma predeterminada en IBM MQ classes for JMS, puede afectar a la velocidad a la que se pueden reutilizar los puertos. Como resultado de ello, se puede producir un error de conexión mientras se liberan puertos.

Compresión de canal en IBM MQ classes for JMS
Una aplicación de IBM MQ classes for JMS puede utilizar recursos de IBM MQ para comprimir la cabecera o los datos de un mensaje.

La compresión de los datos que circulan por un canal de IBM MQ puede mejorar el rendimiento del canal y reducir el tráfico de la red. Mediante una función suministrada con IBM MQ, puede comprimir los datos que circulan por los canales de mensajes y canales de MQI. En ambos tipos de canal, puede comprimir los datos de cabecera y los datos del mensaje de forma separada. De forma predeterminada, no se comprime ningún dato en un canal.

Una aplicación de IBM MQ classes for JMS especifica las técnicas que se pueden utilizar para comprimir los datos de cabecera o de mensaje en una conexión creando un objeto java.util.Collection. Cada técnica de compresión es un objeto Integer de la colección y el orden en que la aplicación añade las técnicas de compresión a la colección es el orden en que se negocian las técnicas de compresión con el gestor de colas cuando la aplicación crea la conexión. A continuación, la aplicación puede pasar la colección al objeto ConnectionFactory invocando el método setHdrCompList(), para datos de cabecera, o el método setMsgCompList(), para datos de mensaje. Cuando la aplicación está preparada, puede crear la conexión.

Los fragmentos de código siguientes muestran el método descrito. El primer fragmento de código muestra cómo implementar la compresión de datos de cabecera:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

El segundo fragmento de código muestra cómo implementar la compresión de datos de mensaje:

```
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();
```

En el segundo ejemplo, las técnicas de compresión se negocian en el orden RLE y, a continuación, ZLIBHIGH, cuando se crea la conexión. La técnica de compresión que se selecciona no se puede modificar durante la duración de un objeto Connection. Para utilizar la compresión en una conexión, se deben invocar los métodos setHdrCompList() y setMsgCompList() antes de crear el objeto Connection.

Transferencia asíncrona de mensajes en IBM MQ classes for JMS
Normalmente, cuando una aplicación envía mensajes a un destino, la aplicación tiene que esperar a que el gestor de colas confirme que ha procesado la solicitud. En algunos casos puede mejorar el rendimiento de la mensajería mediante la transferencia asíncrona de mensajes. Cuando una aplicación transfiere un mensaje asíncronamente, el gestor de colas no devuelve la confirmación de éxito o error de cada llamada, pero el usuario puede comprobar periódicamente la existencia de errores.

El que un destino devuelva el control a la aplicación, sin determinar si el gestor de colas ha recibido el mensaje de forma segura, depende de las propiedades siguientes:

La propiedad de destino de JMS PUTASYNCALLOWED (nombre abreviado - PAALD).
PUTASYNCALLOWED controla si las aplicaciones JMS pueden colocar mensajes de forma asíncrona, si esta opción está permitida por la cola o el tema subyacente que representa el destino JMS.

La propiedad de cola o tema IBM MQ DEFPRESP (tipo de respuesta de colocación predeterminado).
DEFPRESP especifica si las aplicaciones que colocan mensajes en la cola, o publican mensajes en el tema, pueden utilizar la funcionalidad de operación de transferencia asíncrona.

En la tabla siguiente se muestran los valores posibles para las propiedades PUTASYNCALLOWED y DEFPRESP, y las combinaciones de valores utilizados para habilitar la funcionalidad de operación de transferencia asíncrona.

Tabla 46. Cómo se combinan las propiedades PUTASYNCALLOWED y DEFPRESP para determinar si los mensajes se colocan en un destino de forma asíncrona

Propiedad de cola de IBM MQ	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**Funcionalidad de puesta en forma asíncrona habilitada**
DEFPRESP=SYNC	Operación de transferencia asíncrona no habilitada	Funcionalidad de puesta en forma asíncrona habilitada	PUTASYNCALLOWED = AS_DEST o AS_Q_DEF o AS_T_DEF
DEFPRESP=ASYNC	Operación de transferencia asíncrona no habilitada	Funcionalidad de puesta en forma asíncrona habilitada	PUTASYNCALLOWED = AS_DEST o AS_Q_DEF o AS_T_DEF

Puede cambiar el comportamiento especificando la propiedad de destino IBM MQ-JMS para que diga "NO" o "YES" tal como se muestra en la tabla, pero también se puede alterar temporalmente para toda la máquina virtual Java utilizando la JVM **SystemProperty** y el valor:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

Para los mensajes enviados en una sesión transaccional, la aplicación determina en última instancia si el gestor de colas ha recibido los mensajes correctamente cuando la aplicación invoca `commit()`.

Si una aplicación envía mensajes persistentes en una sesión transaccional y uno o más de los mensajes no se reciben correctamente, la transacción no se puede confirmar y genera una excepción. Pero si una aplicación envía mensajes no persistentes en una sesión transaccional y uno o más de los mensajes no se reciben correctamente, la transacción se confirma satisfactoriamente. La aplicación no recibe ninguna respuesta para indicar que los mensajes no persistentes no han llegado correctamente.

Para los mensajes no persistentes enviados en una sesión no transaccional, la propiedad SENDCHECKCOUNT del objeto *ConnectionFactory* especifica cuántos mensajes se deben enviar antes de que IBM MQ classes for JMS compruebe que el gestor de colas ha recibido los mensajes correctamente.

Si una comprobación descubre que uno o más mensajes no se han recibido correctamente y la aplicación ha registrado un escucha de excepción en la conexión, IBM MQ classes for JMS invoca el método `onException()` del escucha de excepción para pasar una excepción de JMS a la aplicación.

La excepción de JMS tiene el código de error JMSWMQ0028 y este código muestra el mensaje siguiente:

```
At least one asynchronous put message failed or gave a warning.
```

La excepción de JMS también tiene una excepción enlazada que proporciona más detalles. El valor predeterminado de la propiedad SENDCHECKCOUNT es cero, lo que significa que no se realizan comprobaciones de este tipo.

Esta optimización es más útil para una aplicación que se conecta a un gestor de colas en la modalidad de cliente, y necesita enviar una secuencia de mensajes en rápida sucesión, pero no necesita una respuesta inmediata del gestor de colas para cada mensaje enviado. Pero una aplicación puede todavía utilizar esta optimización aunque se conecte al gestor de colas en la modalidad de enlaces, pero la mejora de rendimiento prevista no es tan grande.

Nota: Si está utilizando un **`MessageProducer`** no identificado para enviar un mensaje bajo una transacción, de forma predeterminada los mensajes se colocan en la cola utilizando el mecanismo de colocación asíncrona.

Esto puede ocurrir porque la API de JMS permite que se cree el **`MessageProducer`** sin especificar un Destino, utilizando la sintaxis:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

En este escenario, el destino JMS se proporciona cuando se envía el mensaje en lugar de antes de tiempo cuando se construye el **`MessageProducer`** . En términos de la API IBM MQ , esto hace que se emita una MQPUT1 para transferir el mensaje a la cola.

Si lo hace bajo un punto de sincronización IBM MQ , lo que significa (en terminología de JMS) colocar el mensaje bajo una transacción, ya sea utilizando una sesión de JMS transaccionada o mediante el uso de una XA`Session` , la API de IBM MQ classes for JMS cambia a utilizar la colocación asíncrona.

Utilización de la lectura anticipada con IBM MQ classes for JMS
La función de lectura anticipada que proporciona IBM MQ permite que se envíen mensajes no persistentes que se reciben fuera de una transacción a IBM MQ classes for JMS antes de que una aplicación los solicite. Las IBM MQ classes for JMS almacenan los mensajes en un almacenamiento intermedio interno y pasan los mensajes a la aplicación cuando la aplicación los solicita.

Las aplicaciones de IBM MQ classes for JMS que utilizan `MessageConsumers` o `MessageListeners` para recibir mensajes de un destino fuera de una transacción pueden utilizar la función de lectura anticipada. El uso de la lectura anticipada permite que las aplicaciones que utilizan dichos objetos se beneficien de un rendimiento mejorado cuando reciben mensajes.

Una aplicación que utilice `MessageConsumers` o `MessageListeners` podrá utilizar la lectura anticipada en función de las propiedades siguientes:

La propiedad de destino JMS READAHEADALLOWED (nombre abreviado - RAALD).
READAHEADALLOWED controla si las aplicaciones JMS pueden utilizar la lectura anticipada al obtener o examinar mensajes no persistentes fuera de una transacción, si la cola o el tema subyacente que representa el destino JMS, permite esta opción.

La propiedad de tema o cola IBM MQ DEFREADA (lectura anticipada predeterminada).
DEFREADA especifica si las aplicaciones que reciben o examinan mensajes no persistentes fuera de una transacción pueden utilizar la lectura anticipada.

En la tabla siguiente se muestran los valores posibles para las propiedades READAHEADALLOWED y DEFREADA, y las combinaciones de valores utilizadas para habilitar la funcionalidad de lectura anticipada.

Tabla 47. Cómo se combinan las propiedades READAHEADALLOWED y DEFREADA para determinar si se utiliza la lectura anticipada al recibir o examinar mensajes no persistentes fuera de una transacción.

Propiedad de cola de IBM MQ	READAHEADALLOWED = YES	READAHEADALLOWED = NO	AS_DEST o AS_Q_DEF o AS_T_DEF
DEFREADA = NO	Función de lectura anticipada habilitada	Función de lectura anticipada no habilitada	Función de lectura anticipada no habilitada
DEFREADA = YES	Función de lectura anticipada habilitada	Función de lectura anticipada no habilitada	Función de lectura anticipada habilitada
DEFREADA = DISABLED	Función de lectura anticipada no habilitada	Función de lectura anticipada no habilitada	Función de lectura anticipada no habilitada

Si se habilita la función de lectura anticipada, cuando una aplicación crea un `MessageConsumer` o `MessageListener`, las IBM MQ classes for JMS crean un almacenamiento intermedio interno para el destino que está supervisando el `MessageConsumer` o `MessageListener`. Hay un almacenamiento intermedio interno para cada `MessageConsumer` o `MessageListener`. El gestor de colas comienza a enviar mensajes no persistentes a las IBM MQ classes for JMS cuando la aplicación llama a uno de los métodos siguientes:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

Las IBM MQ classes for JMS devuelven automáticamente el primer mensaje a la aplicación, mediante la llamada a método que haya realizado la aplicación. Las IBM MQ classes for JMS almacenan los demás mensajes no persistentes en el almacenamiento intermedio interno que se ha creado para el destino. Cuando la aplicación solicita el siguiente mensaje para su proceso, las IBM MQ classes for JMS devolverán el siguiente mensaje del almacenamiento intermedio interno.

Las IBM MQ classes for JMS solicitan más mensajes no persistentes del gestor de colas cuando el almacenamiento intermedio interno está vacío.

El almacenamiento intermedio interno utilizado por las IBM MQ classes for JMS se suprime cuando una aplicación cierra un `MessageConsumer`, o la sesión de JMS con la que está asociado un `MessageListener`.

Para `MessageConsumers`, los mensajes sin procesar del almacenamiento intermedio interno se pierden.

Al utilizar `MessageListeners`, lo que ocurre con los mensajes del almacenamiento intermedio interno dependerá de la propiedad de destino JMS READAHEADCLOSEPOLICY (nombre abreviado: RACP). El valor predeterminado de la propiedad es DELIVER_ALL, que significa que la sesión JMS que se ha utilizado para crear el `MessageListener` no se cierra hasta que todos los mensajes del almacenamiento intermedio interno se entreguen a la aplicación. Si la propiedad se establece en DELIVER_CURRENT, la sesión JMS se cerrará después de que la aplicación haya procesado el mensaje actual y todos los mensajes restantes del almacenamiento intermedio interno se descartan.

Publicaciones retenidas en IBM MQ classes for JMS
Un cliente de IBM MQ classes for JMS se puede configurar para que utilice publicaciones retenidas.

Una aplicación de publicación puede especificar que se debe retener una copia de una publicación para que se pueda enviar a los futuros suscriptores que muestren un interés en el tema. Esto se realiza en IBM MQ classes for JMS estableciendo la propiedad de entero JMS_IBM_RETAIN en el valor 1. Se han definido constantes para estos valores en la interfaz com.ibm.msg.client.jms.JmsConstants . Por ejemplo, si ha creado un mensaje *msg*, para establecerlo como publicación retenida utilice el código siguiente:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

Ahora puede enviar el mensaje de la forma habitual. También se puede consultar JMS_IBM_RETAIN en un mensaje recibido. Por lo tanto es posible consultar si un mensaje recibido es una publicación retenida.

Soporte de XA en IBM MQ classes for JMS
JMS da soporte a transacciones compatibles con XA en las modalidades de enlaces y de cliente con un gestor de transacciones soportado dentro de un contenedor de JEE.

Si necesita utilizar funciones de XA en un entorno de servidor de aplicaciones, debe configurar la aplicación debidamente. Consulte la documentación de su servidor de aplicaciones para conocer cómo configurar aplicaciones para utilizar transacciones distribuidas.

Un gestor de colas de IBM MQ no puede actuar como gestor de transacciones para JMS.

Retardo de entrega para mensajes JMS
Para JMS 2.0 o posterior, puede especificar un retardo de entrega al enviar un mensaje. El gestor de colas no entregará el mensaje mientras no haya transcurrido el retardo de entrega especificado.

Una aplicación puede especificar un retardo de entrega en milisegundos, cuando envía un mensaje, utilizando `MessageProducer.setDeliveryDelay(long deliveryDelay)` o `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Este valor se añade a la hora de envío del mensaje y proporciona la hora más temprana a la que cualquier otra aplicación puede obtener dicho mensaje.

El retardo de entrega se implementa utilizando una única cola de transferencia interna. Los mensajes que tienen un retardo de entrega distinto de cero se colocan en esta cola con una cabecera que indica el retardo de entrega y la información relativa a la cola de destino. Un componente del gestor de colas llamado procesador de retardos de entrega supervisa los mensajes en la cola de transferencia. Cuando se completa el retardo de entrega de un mensaje, este se saca de la cola de transferencia y se coloca en la cola de destino.

Clientes de mensajería

La implementación de IBM MQ del retardo de entrega solo está disponible para su uso en el cliente JMS. Las restricciones siguientes se aplican si está utilizando el retardo de entrega con IBM MQ. Estas restricciones se aplican igualmente a `MessageProducers` y `JMSProducers`, pero `JMSRuntimeExceptions` se generan en el caso de `JMSProducers`.

- Cualquier intento de llamar a `MessageProducer.setDeliveryDelay` con un valor distinto de cero cuando se conecta a un gestor de colas anterior a IBM MQ 8.0, da como resultado un `JMSException` con un mensaje MQRC_FUNCTION_NOT_SUPPORTED.
- El retardo de entrega no está soportado en destinos en clúster que tienen un valor **DEFBIND** distinto de MQBND_BIND_NOT_FIXED. Si un `MessageProducer` tiene configurado un retardo de entrega distinto de cero y se intenta enviar a un destino que no cumple este requisito, la llamada dará una `JMSException` con un mensaje MQRC_OPTIONS_ERROR.
- Cualquier intento de establecer un valor de tiempo de vida inferior a un retardo de entrega distinto de cero especificado anteriormente, o viceversa, da una `JMSException` con un mensaje MQRC_EXPIRY_ERROR. Esta comprobación se realiza al invocar los métodos `setTimeToLive`, `setDeliveryDelay` o `send`, dependiendo del conjunto exacto de operaciones seleccionadas.

- El uso de las publicaciones retenidas y el retardo de entrega no están soportados. Si se intenta publicar un mensaje con un retardo de entrega cuando dicho mensaje se ha marcado como retenido utilizando `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` da una `JMSException` con un mensaje MQRC_OPTIONS_ERROR.
- El retardo de entrega y la agrupación de mensajes no están soportados y cualquier intento de utilizar esta combinación da como resultado un `JMSException` con un mensaje MQRC_OPTIONS_ERROR.

Cualquier error al enviar un mensaje con retraso de entrega da lugar a que el cliente lance una `JMSException` con el correspondiente mensaje de error, por ejemplo, cola llena. En algunas situaciones, es posible que el mensaje de error se aplique al destino objetivo o a la cola de transferencia, o a ambos.

Nota: IBM MQ permite que las aplicaciones que colocan un mensaje en una unidad de trabajo vuelvan a obtener el mismo mensaje, aunque la unidad de trabajo no se haya confirmado. Esta técnica no funciona con retardo de entrega, ya que el mensaje no se coloca en la cola de transferencia mientras no se confirma la unidad de trabajo y, por tanto, no habrá sido enviado al destino objetivo.

Autorización

IBM MQ realiza comprobaciones de autorización en el destino objetivo original cuando la aplicación envía un mensaje con un retardo de entrega distinto de cero. Si la aplicación no está autorizada, el envío fallará. Cuando el gestor de colas detecta que el retardo de entrega de un mensaje se ha completado, abre la cola de destino. No se realizan comprobaciones de autorización en este punto.

SYSTEM.DDELAY.LOCAL.QUEUE

Una cola del sistema, SYSTEM.DDELAY.LOCAL.QUEUE, se utiliza para implementar el retardo de entrega.

- Multi En Multiplatforms, SYSTEM.DDELAY.LOCAL.QUEUE existe de forma predeterminada. Hay que modificar la cola de sistema para que sus atributos MAXMSGL y MAXDEPTH sean suficientes para la carga esperada.
- z/OS En IBM MQ for z/OS, SYSTEM.DDELAY.LOCAL.QUEUE se utiliza como cola de transferencia para los mensajes que se envían con retardo de entrega a colas locales y compartidas. En z/OS, la cola debe crearse y definirse para que sus atributos MAXMSGL y MAXDEPTH sean suficientes para la carga esperada.

Cuando se crea esta cola, hay que protegerla para que accedan a ella el menor número de usuarios posible. El acceso a la cola solo habrá de tener lugar a efectos de mantenimiento y supervisión.

Cuando un mensaje es enviado por una aplicación JMS con un retardo de entrega distinto a cero, se coloca en esta cola con un nuevo ID de mensaje. El ID de mensaje original se coloca en el ID de correlación del mensaje. Este ID de correlación permite a una aplicación recuperar un mensaje de la cola de transferencia cuando sea necesario, por ejemplo, si por error se ha utilizado un retardo de entrega grande.

Consideraciones sobre z/OS

z/OS

Si el sistema se está ejecutando en z/OS, hay consideraciones adicionales que se deben tener en cuenta, si desea utilizar el retardo de entrega.

Si se va a usar un retardo de entrega, hay que definir la cola de sistema SYSTEM.DDELAY.LOCAL.QUEUE. Debe definirse con una clase de almacenamiento que sea suficiente para su carga esperada, y con INDXTYPE (NONE) y MSGDLVSQ (FIFO) especificados. Se proporciona una definición de ejemplo de la cola de sistema, comentada, en el JCL CSQ4INSG.

Colas compartidas

El retardo de entrega está soportado en el envío de mensajes a colas compartidas. No obstante, solo se usa una única cola de transferencia privada, independientemente de que la cola de destino sea compartida o no. El gestor de colas propietario de dicha cola privada tiene que estar ejecutando para enviar el mensaje retardado a la cola compartida de destino cuando se complete el retardo.

Nota: Si se coloca un mensaje no persistente con un retardo de entrega en una cola compartida y se cierra el gestor de colas propietario de la cola de transferencia, se perderá el mensaje original. Por tanto, es más probable que se pierdan los mensajes no persistentes enviados con retardo de entrega a una cola compartida que los no persistentes enviados sin retardo de entrega a una cola compartida.

Resolución del destino objetivo

Si el mensaje se envía a una cola, la resolución se lleva a cabo dos veces; una vez por parte de la aplicación JMS y una vez por parte del gestor de colas, cuando retira el mensaje de la cola de transferencia y lo envía a la cola de destino.

Las suscripciones de destino para las publicaciones se emparejan cuando la aplicación JMS llama al método de envío.

Si un mensaje se envía con persistencia o prioridad conforme a la definición de cola, el valor se establece en la primera resolución y no en la segunda.

Intervalo de caducidad

El retardo de entrega conserva el comportamiento de la propiedad de caducidad, **MQMD.Expiry**. Por ejemplo, si un mensaje se ha colocado desde una aplicación JMS con un intervalo de caducidad de 20.000 ms y un retardo de entrega de 5.000 ms y se ha obtenido una vez transcurrido un tiempo de 10.000 ms, el valor del campo MQMD.expiry podría ser aproximadamente unas 50 décimas de segundo. Este valor indica que han transcurrido 15 segundos desde el momento en que se ha colocado el mensaje hasta el momento en que se ha recuperado.

Si un mensaje caduca mientras se encuentra en la cola de transferencia y se define una de las opciones MQRO_EXPIRATION_*, el informe generado corresponde al mensaje original tal y como lo envía la aplicación, eliminándose la cabecera utilizada para contener la información de retardo de entrega.

Parada e inicio del procesador de retardos de entrega

z/OS En z/OS, el procesador del retardo de entrega se integra en el espacio de direcciones MSTR del gestor de colas. Cuando se inicia el gestor de colas, también se inicia el procesador de retardos. Si la cola de transferencia está disponible, abre la cola y espera a que le lleguen mensajes para procesarlos. Si la cola de transferencia no está definida o se le inhabilitan las obtenciones, o se produce otro error, el procesador de retardos de entrega se cierra. Si la cola de transferencia se define más adelante o se modifica para habilitarle las obtenciones, se reinicia el procesador de retardos de entrega. Si el procesador de retardo de entrega concluye por cualquier otro motivo, se puede reiniciar alterando el atributo **PUT** de la cola de transferencia de ENABLED a DISABLED y de nuevo a ENABLED . Si fuera necesario parar el procesador de retardos de entrega por cualquier motivo, establezca el atributo PUT de la cola de transferencia a DISABLED.

Multi En Multiplatforms, el procesador de retardos se inicia con el gestor de colas y se reinicia automáticamente en caso de que se produzca un error recuperable.

Fallo al colocar en una cola de destino

Si no se puede colocar un mensaje retardado en la cola de destino una vez que se completa su retardo, dicho mensaje se tratará tal como se indica en sus opciones de informe: o se descarta o se envía a la cola de mensajes no entregados. Si esta acción falla, se intenta colocar el mensaje más tarde. Si la acción es satisfactoria, se genera un informe de excepciones y se envía a la cola especificada, si se solicita el informe. Si no se ha podido enviar el mensaje de informe, dicho mensaje se envía a la cola de mensajes no

entregados. Si falla el envío del informe a la cola de mensajes no entregados y el mensaje es persistente, todos los cambios se descartan y el mensaje original se retrotrae y se vuelve a entregar posteriormente. Si el mensaje no es persistente, se descarta el mensaje de informe, pero se confirmman otros cambios. Si no se puede entregar una publicación retardada porque un suscriptor ha anulado su suscripción, o si el suscriptor no es persistente, porque se ha desconectado, el mensaje se descarta silenciosamente. Los mensajes de informe se siguen generando en la forma descrita anteriormente.

Si una publicación retardada no se puede entregar a un suscriptor y se coloca en la cola de mensajes no entregados, y falla la colocación en la cola de mensajes no entregados , el mensaje se descarta.

Para reducir la probabilidad de que falle la colocación en la cola de destino después de que se haya completado el retardo de entrega, el gestor de colas realiza algunas comprobaciones básicas cuando el cliente JMS envía un mensaje con un retardo de entrega distinto a cero. Estas comprobaciones incluyen si la cola está inhabilitada, si el mensaje es más grande que la longitud máxima de mensaje permitida y si la cola está llena.

Publicación/suscripción

El emparejamiento de una publicación con las suscripciones disponibles se produce cuando la aplicación JMS envía un mensaje con un retardo de entrega distinto de cero. Un mensaje por cada suscriptor coincidente se coloca en la cola SYSTEM.DDELAY.LOCAL.QUEUE, donde se guarda hasta que se completa el retardo de entrega. Si uno de estos suscriptores es una suscripción de proxy de otro gestor de colas, la diseminación (fan-out) en dicho gestor de colas se produce una vez completado el retardo de entrega. Esto podría dar lugar a que un suscriptor del otro gestor de colas reciba publicaciones publicadas antes de suscribirse. Se trata de una desviación de la especificación JMS 2.0 o posterior.

El retardo de entrega con publicación/suscripción solo está soportado si el tema de destino está configurado con (N)PMSGDLV = ALLAVAIL. Un intento de utilizar cualquier otro valor dará un error MQRC_PUBLICATION_FAILURE. Si el procesador de retardos de entrega falla mientras está colocando el mensaje en la cola de destino, el resultado es el descrito en la sección "Fallo al colocar en la cola de destino".

Mensajes de informe

El procesador de entregas soporta y pone en marcha todas las opciones de informe distintas de las siguientes opciones, que se ignoran, pero que se pasan en el mensaje cuando se envía a la cola de destino:

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

Suscripciones clonadas y compartidas

En IBM MQ 8.0 o posterior, hay dos métodos para otorgar a varios consumidores acceso a la misma suscripción. Estos dos métodos son mediante suscripciones clonadas o mediante suscripciones compartidas.

Suscripciones clonadas

La suscripción clonada es una extensión de IBM MQ. Las suscripciones clonadas permiten a varios consumidores en diferentes máquinas virtuales Java (JVM) acceder simultáneamente a la suscripción. Este comportamiento se puede utilizar estableciendo la propiedad **CLONESUPP** en `Enabled` en un objeto connectionFactory. De forma predeterminada, **CLONESUPP** es `Disabled`. Las suscripciones clonadas solo pueden habilitarse en suscripciones duraderas. Si **CLONESUPP** está habilitado, cada conexión posterior que se realice utilizando esta connectionFactory se clona.

Una suscripción duradera puede considerarse clonada si se crean uno o varios consumidores para recibir mensajes de dicha suscripción, es decir, si se crean especificando el mismo nombre de suscripción.

Esto solo puede hacerse si la conexión con la que se han creado los consumidores tiene **CLONESUPP** establecido en Enabled en MQConnectionFactory. Cuando se publica un mensaje en el tema de una suscripción, se envía una copia de ese mensaje a la suscripción. El mensaje está disponible para todos los consumidores, pero solo uno lo recibe.

Nota: La habilitación de las suscripciones clonadas amplía la especificación JMS

Suscripciones compartidas

Las suscripciones compartidas, introducidas por la especificación JMS 2.0 , permiten que los mensajes de una suscripción de tema se compartan entre varios consumidores. Cada mensaje de la suscripción se entrega solo a uno de los consumidores de esa suscripción. Las suscripciones compartidas se habilitan mediante la llamada relevante a la API JMS 2.0 o posterior.

Las API pueden invocarse de varias formas:

- Desde una aplicación Java SE (o contenedor de cliente Java EE)
- Desde un servlet o la implementación de un MDB.

La especificación JMS 2.0 o posterior no define ninguna forma estándar de dirigir un MDB desde una suscripción compartida, por lo que IBM MQ 8.0 o posterior proporciona la propiedad de especificación de activación **sharedSubscription** para esta finalidad. Para obtener más información sobre esta propiedad, consulte "Configuración del adaptador de recursos para la comunicación de entrada" en la página 465 y "Ejemplos de cómo definir la propiedad sharedSubscription" en la página 484.

Si se habilita una suscripción compartida, no puede no compartirse.

Las suscripciones compartidas pueden crearse como suscripciones duraderas o no duraderas. No es necesario crear por separado objetos en el lado del gestor de colas más allá de la configuración normal de JMS . Los objetos necesarios se crean dinámicamente.

Cómo decidir entre suscripciones compartidas o clonadas

Cuando decida si desea utilizar suscripciones compartidas o clonadas, tenga en cuenta las ventajas de ambas. Siempre que sea posible, utilice suscripciones compartidas, ya que es un comportamiento definido por la especificación, en lugar de una extensión específica de IBM MQ .

La tabla siguiente contiene algunos de los puntos que se deben tener en cuenta para decidir entre suscripciones compartidas y clonadas:

Tabla 48. Comparación de consideraciones para suscripciones compartidas y suscripciones clonadas

Suscripciones compartidas	**Suscripciones clonadas**
Las suscripciones compartidas son una parte estándar de la especificación JMS 2.0 o posterior.	Las suscripciones clonadas son una extensión específica de IBM MQ.
Las suscripciones compartidas se crean utilizando llamadas de métodos de API explícitas.	Las suscripciones clonadas se controlan administrativamente a nivel de ConnectionFactory.
Las suscripciones compartidas pueden ser duraderas o no duraderas.	Las suscripciones clonadas solo pueden ser duraderas.
Las suscripciones compartidas se crean explícitamente para cada suscripción individual.	Las suscripciones clonadas se utilizan para cualquier suscripción duradera con una conexión para la que se ha habilitado la función.
Si se crea una suscripción como compartida, no se puede cambiar posteriormente a no compartida, y viceversa.	Una suscripción puede cambiarse de clonada a no clonada cada vez que se vuelve a abrir si cambia la propiedad **CLONESUPP** de la conexión propietaria.

Conceptos relacionados

Suscriptores y suscripciones

Durabilidad de suscripción

Tareas relacionadas

Utilización de suscripciones compartidas de JMS 2.0

Referencia relacionada

“Ejemplos de cómo definir la propiedad sharedSubscription” en la página 484
Puede definir la propiedad sharedSubscription de una especificación de activación dentro de un archivo WebSphere Liberty `server.xml` . Como alternativa, también puede definir la propiedad dentro de un bean controlado por mensaje (MDB) utilizando anotaciones.

CLONESUPP

Propiedad SupportMQExtensions

La especificación JMS 2.0 ha introducido cambios en la forma en que funcionan determinados comportamientos. IBM MQ 8.0 y posteriores incluyen la propiedad **`com.ibm.mq.jms.SupportMQExtensions`**, que se puede establecer en TRUE para revertir estos comportamientos cambiados a implementaciones anteriores.

V 9.3.0 JM 3.0 V 9.3.0 La propiedad **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) está soportada por IBM MQ classes for Jakarta Messaging, que están disponibles en `com.ibm.mq.jakarta.client.jar`.

JMS 2.0 La propiedad **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) está soportada por IBM MQ classes for JMS, que están disponibles en `com.ibm.mq.allclient.jar` o `com.ibm.mqjms.jar`.

Hay tres áreas de funcionalidad que se revierten estableciendo **`SupportMQExtensions`** en `True`:

Prioridad de mensaje
: A los mensajes se les puede asignar una prioridad de 0 a 9. Antes de JMS 2.0, los mensajes también podían utilizar el valor -1, lo que indica que se utiliza la prioridad predeterminada de una cola. JMS 2.0 y posteriores no permiten que se establezca una prioridad de mensaje de -1. Activar **`SupportMQExtensions`** permite utilizar el valor de -1.

ID de cliente
: La especificación JMS 2.0 o posterior requiere que se compruebe la exclusividad de los ID de cliente no nulos cuando realizan una conexión. Activar **`SupportMQExtensions`** significa que este requisito no se tiene en cuenta y que se puede reutilizar un ID de cliente.

NoLocal
: La especificación JMS 2.0 o posterior requiere que cuando se activa esta constante, un consumidor no puede recibir mensajes publicados por el mismo ID de cliente. Antes de JMS 2.0, este atributo estaba establecido en un suscriptor para evitar la recepción mensajes publicados por su propia conexión. La activación de **`SupportMQExtensions`** revierte este comportamiento a su implementación anterior.

Esta propiedad se puede establecer de la forma siguiente:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Esta propiedad se puede establecer como propiedad JVM estándar en el mandato **`java`** o incluida dentro del archivo de configuración de IBM MQ classes for JMS.

Conceptos relacionados

“El archivo de configuración IBM MQ classes for JMS/Jakarta Messaging” en la página 102
Los archivos de configuración IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging especifican propiedades que se utilizan para configurar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging.

Referencia relacionada

“Propiedades utilizadas para configurar el comportamiento del cliente JMS” en la página 109
Utilice estas propiedades para configurar el comportamiento del cliente JMS.

Utilización de suscripciones compartidas en aplicaciones JMS

Con las suscripciones compartidas, una sola suscripción se comparte entre varios consumidores, con solo uno de los consumidores recibiendo una publicación en cualquier momento.

Las suscripciones compartidas están disponibles desde JMS 2.0 en adelante. Al desarrollar una aplicación JMS para IBM MQ 8.0 o posterior, es posible que tenga que tener en cuenta el impacto de esta funcionalidad en el gestor de colas.

La idea detrás de las suscripciones compartidas es compartir la carga entre varios consumidores. Una suscripción duradera también se puede compartir entre varios consumidores.

Por ejemplo, suponga que hay:

- Una suscripción SUB, que está suscrita a un tema FIFA2014/UPDATES para recibir actualizaciones de partidos de futbol, que es compartida por tres consumidores: C1, C2 y C3
- Un productor P1 que publica en el tema FIFA2014/UPDATES

Cuando se realiza una publicación en FIFA2014/UPDATES, la publicación solo será recibida por uno de los tres consumidores (C1 o C2 o C3) pero no por todos.

El ejemplo siguiente muestra el uso de suscripciones compartidas y también muestra el uso de la API adicional en JMS 2.0, `Message.receiveBody()`, para recuperar sólo el cuerpo del mensaje.

El ejemplo crea tres hebras de suscriptor, que crean una suscripción compartida al tema FIFA2014/UPDATES y una hebra de publicador.

V 9.3.0 JM 3.0 V 9.3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");
```

```
            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);
```

```
                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
```

```
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

Conceptos relacionados

Interfaces de lenguaje de IBM MQ Java

V 9.3.2 Configuración de la aplicación modular para utilizar IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging

V 9.3.2 Puede utilizar IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging de forma modular requiriendo el módulo adecuado dentro de la aplicación e incluyendo el directorio adecuado en la vía de acceso del módulo.

El embalaje modular

Los archivos JAR unificados para IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging proporcionan nombres de módulo automáticos, que sustituyen a los nombres predeterminados que se derivan de los nombres de archivo JAR.

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) se proporcionan con un nombre de módulo de `com.ibm.mq.javax`.
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) se proporcionan con un nombre de módulo de `com.ibm.mq.jakarta`.

El directorio `MQ_HOME/java/lib` predeterminado no es adecuado para el uso modular porque los módulos no pueden contener el mismo paquete y el directorio predeterminado contiene los mismos paquetes en varios JAR. Por lo tanto, hay nuevos directorios disponibles para que contengan sólo los archivos JAR necesarios, sin duplicación de paquetes entre los JAR. Estos directorios son adecuados para su inclusión en un `module-path`.

Nota: Si tiene aplicaciones que utilizan los archivos JAR disponibles en un contexto modular basándose en los nombres de módulo predeterminados, debe actualizar las aplicaciones para que requieran los nuevos nombres de módulo. Los nombres de módulo predeterminados se derivan de los nombres de archivo JAR.

Configuración de la aplicación modular para utilizar IBM MQ classes for JMS

Puede configurar la aplicación modular para que utilice IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) realizando los pasos siguientes:

- Configure la aplicación para que requiera el módulo `com.ibm.mq.javax` .
- Configure la aplicación para incluir el directorio `MQ_HOME/java/lib/modules/javax` en la vía de acceso del módulo.

Configuración de la aplicación modular para utilizar IBM MQ classes for Jakarta Messaging

Puede configurar la aplicación modular para que utilice IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) realizando los pasos siguientes:

- Configure la aplicación para que requiera el módulo `com.ibm.mq.jakarta` .
- Configure la aplicación para incluir el directorio `MQ_HOME/java/lib/modules/jakarta` en la vía de acceso del módulo.

Configuración de la aplicación modular para utilizar IBM MQ classes for Java

Para utilizar IBM MQ classes for Java desde una aplicación modular, puede utilizar la configuración para IBM MQ classes for JMS o la configuración para IBM MQ classes for Jakarta Messaging, ya que ambos archivos JAR de cliente dan soporte a IBM MQ classes for Java. Sin embargo, la aplicación sólo debe utilizar una de estas configuraciones, no ambas.

Recursos del servidor de aplicaciones de IBM MQ classes for JMS

Este tema describe cómo IBM MQ classes for JMS implementa la clase ConnectionConsumer y funcionalidad avanzada en la clase Session. También describe la función de una agrupación de sesiones de servidor.

Importante: Esta información es únicamente de referencia. No se debe escribir una aplicación para utilizar esta interfaz: se utiliza dentro del adaptador de recursos de IBM MQ para conectar con servidores de Java EE. Para obtener información de conexión práctica, consulte “Utilización del adaptador de recursos de IBM MQ” en la página 448.

IBM MQ classes for JMS da soporte a ASF (Application Server Facilities) que se especifican en la *Especificación de Java Message Service* (consulte Oracle Technology Network for Java Developers). Esta especificación identifica tres funciones dentro de este modelo de programación:

- **El proveedor JMS** proporciona ConnectionConsumer y una funcionalidad de sesión avanzada.
- **El servidor de aplicaciones**, que proporciona las funciones ServerSessionPool y ServerSession.
- **La aplicación cliente**, que utiliza la funcionalidad proporcionada por el proveedor de JMS y el servidor de aplicaciones.

La información de este tema no es aplicable si una aplicación utiliza una conexión en tiempo real con un intermediario.

La interfaz ConnectionConsumer de JMS

La interfaz ConnectionConsumer proporciona un método de alto rendimiento para entregar mensajes de forma simultánea a una agrupación de hebras.

La especificación de JMS permite que un servidor de aplicaciones se integre estrechamente con una implementación de JMS utilizando la interfaz `ConnectionConsumer`. Esta característica proporciona un proceso simultáneo de mensajes. Normalmente, un servidor de aplicaciones crea una agrupación de hebras, y la implementación de JMS hace que los mensajes estén disponibles para estas hebras. Un servidor de aplicaciones compatible con JMS (tal como WebSphere Application Server) puede utilizar esta característica para proporcionar funciones de mensajería de alto nivel, tales como beans controlados por mensaje.

Las aplicaciones normales no utilizan ConnectionConsumer, pero los clientes expertos de JMS pueden utilizarlo. Para este tipo de clientes, ConnectionConsumer proporciona un método de alto rendimiento para entregar mensajes simultáneamente a una agrupación de hebras. Cuando un mensaje llega a una cola o a un tema, JMS selecciona una hebra de la agrupación y le entrega un lote de mensajes. Para hacer esto, JMS ejecuta un método asociado `onMessage()` de MessageListener.

Puede obtener el mismo resultado construyendo varios objetos Session y MessageConsumer, cada uno de ellos con un MessageListener registrado. Sin embargo, ConnectionConsumer ofrece mejor rendimiento, menor utilización de recursos, mayor flexibilidad y, en especial, se requieren menos objetos Session.

Planificación de una aplicación con ASF

Esta sección describe cómo planificar una aplicación, e incluye:

- “Principios generales de la mensajería punto a punto mediante ASF” en la página 343
- “Principios generales de la mensajería de publicación/suscripción utilizando ASF” en la página 344
- “Eliminación de mensajes de la cola en ASF” en la página 345
- Manejo de mensajes no entregables en ASF. Consulte “Manejo de mensajes no entregables en IBM MQ classes for JMS” en la página 240.

Principios generales de la mensajería punto a punto mediante ASF

Este tema proporciona información general sobre la mensajería punto a punto mediante ASF.

Cuando una aplicación crea un ConnectionConsumer a partir de un objeto QueueConnection, especifica un objeto de cola de JMS y una serie de selector. A continuación, ConnectionConsumer empieza a proporcionar mensajes a las sesiones de la ServerSessionPool asociada. Los mensajes llegan a la cola y, si coinciden con el selector, se entregan a las sesiones de la ServerSessionPool asociada.

En términos de IBM MQ, el objeto de cola hace referencia a un QLOCAL o a un QALIAS en el gestor de colas local. Si es un QALIAS, ese QALIAS debe hacer referencia a un QLOCAL. El QLOCAL de IBM MQ totalmente resuelto se conoce como *QLOCAL subyacente*. Un ConnectionConsumer se dice que está *activo* si no está cerrado y su QueueConnection padre está iniciado.

Varios ConnectionConsumers, cada uno de ellos con selectores diferentes, se pueden ejecutar para el mismo QLOCAL subyacente. Para mantener el rendimiento, los mensajes no deseados no se deben acumular en la cola. Los mensajes no deseados son aquellos para los que no hay ningún ConnectionConsumer activo que tenga un selector coincidente. Puede establecer

QueueConnectionFactory de modo que los mensajes no deseados se eliminen de la cola (para conocer detalles, consulte "Eliminación de mensajes de la cola en ASF" en la página 345). Para establecer este comportamiento en una de estas dos maneras:

- Utilice la herramienta de administración de JMS para establecer QueueConnectionFactory en MRET(NO).
- En el programa, utilice:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Si no cambia este valor, el comportamiento predeterminado es retener los mensajes no deseados en la cola.

Cuando configure el gestor de colas de IBM MQ, tenga en cuenta lo siguiente:

- El QLOCAL subyacente debe estar habilitado para entrada compartida. Para ello, utilice el mandato MQSC siguiente:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- El gestor de colas debe tener una cola de mensajes no entregados habilitada. Si un ConnectionConsumer experimenta algún problema al colocar un mensaje en la cola de mensajes no entregados, la entrega de mensajes del QLOCAL subyacente se detiene. Para definir una cola de mensajes no entregados, utilice:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- El usuario que ejecuta ConnectionConsumer debe tener autorización para realizar MQOPEN con MQOO_SAVE_ALL_CONTEXT y MQOO_PASS_ALL_CONTEXT. Para conocer detalles, consulte la documentación de IBM MQ correspondiente a la plataforma que utilice.
- Cuando los mensajes no deseados se dejan en la cola, reducen el rendimiento del sistema. Por ello, planifique los selectores de mensaje de modo que, entre ellos, los ConnectionConsumers eliminen todos los mensajes de la cola.

Para obtener detalles sobre los mandatos MQSC, consulte Mandatos MQSC.

Principios generales de la mensajería de publicación/suscripción utilizando ASF
Los ConnectionConsumers reciben mensajes para un tema especificado. Un ConnectionConsumer puede ser duradero o no duradero. Es necesario especificar la o las colas utilizadas por el ConnectionConsumer.

Cuando una aplicación crea un ConnectionConsumer desde un objeto TopicConnection, especifica un objeto Topic y una serie de selector. A continuación, ConnectionConsumer empieza a recibir mensajes que coinciden con el selector para ese tema, incluidas las publicaciones retenidas para el tema suscrito.

Como alternativa, una aplicación puede crear un ConnectionConsumer duradero que esté asociado a un nombre específico. ConnectionConsumer recibe los mensajes que se han publicado sobre el tema desde la última vez que ha estado activo el ConnectionConsumer duradero. Recibe todos los mensajes sobre el tema que coinciden con el selector. Pero si ConnectionConsumer utiliza la lectura anticipada, puede perder mensajes no persistentes que se encuentran en el almacenamiento intermedio del cliente cuando el cliente se cierra.

Si IBM MQ classes for JMS está en la modalidad de migración del proveedor de mensajería de IBM MQ, se utiliza una cola separada para las suscripciones de ConnectionConsumer no duraderas.La opción configurable CCSUB de TopicConnectionFactory especifica la cola que se va a utilizar. Normalmente, CCSUB especifica una sola cola para que la utilicen todos los ConnectionConsumers que usan la misma TopicConnectionFactory. Pero es posible hacer que cada ConnectionConsumer genere una cola temporal especificando un prefijo de nombre de cola seguido de un asterisco (*).

Si IBM MQ classes for JMS está en la modalidad de migración del proveedor de mensajería de IBM MQ, la propiedad CCDSUB del tema especifica la cola que se debe utilizar para suscripciones duraderas. De

nuevo, puede ser una cola que ya existe o un prefijo de nombre de cola seguido de un asterisco (*). Si especifica una cola que ya existe, todos los ConnectionConsumers duraderos que se suscriben al tema utilizan esta cola. Si especifica un prefijo de nombre de cola seguido de un asterisco (*), se genera una cola la primera vez que se crea un ConnectionConsumer duradero con un nombre determinado. Esta cola se vuelve a utilizar posteriormente cuando se crea un ConnectionConsumer duradero con el mismo nombre.

Cuando configure el gestor de colas de IBM MQ, tenga en cuenta lo siguiente:

- El gestor de colas debe tener una cola de mensajes no entregados habilitada. Si un ConnectionConsumer experimenta algún problema al colocar un mensaje en la cola de mensajes no entregados, la entrega de mensajes del QLOCAL subyacente se detiene. Para definir una cola de mensajes no entregados, utilice:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- El usuario que ejecuta ConnectionConsumer debe tener autorización para realizar MQOPEN con MQOO_SAVE_ALL_CONTEXT y MQOO_PASS_ALL_CONTEXT. Para obtener detalles, consulte la documentación de IBM MQ correspondiente a la plataforma utilizada.
- Puede optimizar el rendimiento de un ConnectionConsumer individual creando una cola dedicada separada para él. Esto es a expensas de una mayor utilización de los recursos.

Eliminación de mensajes de la cola en ASF
Cuando una aplicación utiliza ConnectionConsumers, JMS puede necesitar eliminar mensajes de la cola en varias situaciones.

Estas situaciones son las siguientes:

Mensaje con formato incorrecto
JMS no puede analizar el mensaje recibido.

Mensaje no entregable
Un mensaje puede alcanzar el umbral de restitución, pero ConnectionConsumer no puede colocarlo en la cola de retirada.

ConnectionConsumer no interesado
Para la mensajería punto a punto, cuando QueueConnectionFactory se establece de modo que no retenga los mensajes no deseados, puede llegar un mensaje que ningún ConnectionConsumer desee.

En estas situaciones, ConnectionConsumer intenta eliminar el mensaje de la cola. Las opciones de disposición contenidas en el campo de informe de la MQMD del mensaje definen el comportamiento exacto. Estas opciones son las siguientes:

MQRO_DEAD_LETTER_Q
El mensaje se coloca en la cola de mensajes no entregados del gestor de colas. Este es el valor predeterminado.

MQRO_DISCARD_MSG
El mensaje se descarta.

ConnectionConsumer también genera un mensaje de informe, que también depende del campo de informe de la MQMD del mensaje. Este mensaje se envía a ReplyToQ del mensaje en ReplyToQmgr. Si se produce un error durante el envío del mensaje de informe, el mensaje se envía a la cola de mensajes no entregados. Las opciones de informe de excepción contenidas en el campo de informe de la MQMD del mensaje establecen los detalles del mensaje de informe. Estas opciones son las siguientes:

MQRO_EXCEPTION
Se genera un mensaje de informe que contiene la MQMD del mensaje original. No contiene ningún dato de cuerpo del mensaje.

MQRO_EXCEPTION_WITH_DATA
Se genera un mensaje de informe que contiene la MQMD, todas las cabeceras MQ y 100 bytes de datos del cuerpo.

MQRO_EXCEPTION_WITH_FULL_DATA
Se genera un mensaje de informe que contiene todos los datos del mensaje original.

valor predeterminado
No se genera ningún mensaje de informe.

Cuando se generan mensajes de informe, se aceptan las opciones siguientes:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Si un mensaje no entregable se puede poner de nuevo en cola, quizás debido a que la cola de mensajes no entregados está llena o la autorización no está bien especificada, lo que ocurre depende de la persistencia del mensaje. Si el mensaje es no persistente, se descarta y no se genera ningún mensaje de informe. Si el mensaje es persistente, se detiene la entrega de mensajes a todos los consumidores de conexión que están a la escucha en ese destino. Estos consumidores de conexión deben estar cerrados y el problema estar resuelto para que se puedan volver a crear y reiniciar la entrega de mensajes.

Es importante definir una cola de mensajes no entregados y comprobarla con regularidad para verificar que no se ha producido ningún problema. En especial, debe asegurarse de que la cola de mensajes no entregados no alcanza la capacidad máxima y que el tamaño máximo de mensajes es suficientemente grande para todos los mensajes.

Cuando un mensaje se coloca en la cola de mensajes no entregados, se le añade como prefijo una cabecera de mensaje no entregado de IBM MQ (MQDLH). Consulte MQDLH - Cabecera de mensaje no entregado para conocer detalles sobre el formato de la cabecera MQDLH. Los campos siguientes le permiten identificar los mensajes que un ConnectionConsumer ha colocado en la cola de mensajes no entregados o los mensajes de informe que ha generado:

- PutApplType es `MQAT_JAVA (0x1C)`
- PutApplName es "`MQ JMS ConnectionConsumer`"

Estos campos se encuentran en la cabecera MQDLH de los mensajes contenidos en la cola de mensajes no entregados y en la cabecera MQMD de los mensajes de informe. El campo de información de retorno de MQMD y el campo de Razón de MQDLH contienen un código que describe el error. Para conocer detalles sobre estos códigos, consulte "Códigos de razón y de información de retorno en ASF" en la página 347. Encontrará la descripción de otros campos en MQDLH - Cabecera de mensaje no entregado.

Manejo de mensajes dañados en ASF
Dentro de ASF (Application Server Facilities), los mensajes no entregables se manejan de forma ligeramente diferente a como se manejan en otros lugares de IBM MQ classes for JMS.

Para obtener información sobre el manejo de mensajes no entregables en IBM MQ classes for JMS, consulte "Manejo de mensajes no entregables en IBM MQ classes for JMS" en la página 240.

Al utilizar los Recursos del servidor de aplicaciones (ASF), el ConnectionConsumer, y no MessageConsumer, procesa mensajes dañados. El ConnectionConsumer vuelve a colocar en cola mensajes de acuerdo con las propiedades BackoutThreshold y BackoutRequeueQName de la cola.

Cuando una aplicación utiliza ConnectionConsumers, las circunstancias en las cuales se restituye un mensaje dependen de la sesión que proporciona el servidor de aplicaciones:

- Cuando la sesión es una sesión sin transacción, con AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE, un mensaje solo se restituye después de un error del sistema, o si la aplicación termina de forma inesperada
- Cuando la sesión es no transaccional y utiliza CLIENT_ACKNOWLEDGE, el servidor de aplicaciones puede restituir los mensajes sin acuse de recibo invocando Session.recover().

  Normalmente, la implementación de cliente de MessageListener o el servidor de aplicaciones llama a Message.acknowledge(). Message.acknowledge() reconoce todos los mensajes entregados en la sesión hasta ahora.

- Cuando la sesión es transaccional, el servidor de aplicaciones puede restituir los mensajes sin acuse de recibo invocando Session.rollback().
- Si el servidor de aplicaciones proporciona una XASession, los mensajes se confirman o restituyen dependiendo de una transacción distribuida. El servidor de aplicaciones se encarga de finalizar la transacción.

Conceptos relacionados

"Manejo de mensajes no entregables en IBM MQ classes for JMS" en la página 240
Un mensaje con formato incorrecto es uno que no puede ser procesado por una aplicación receptora. Si se entrega un mensaje con formato incorrecto a una aplicación y se retrotrae un número especificado de veces, las IBM MQ classes for JMS pueden moverlo a una cola de retirada.

Manejo de errores

Esta sección describe diversos aspectos del manejo de errores, incluidos "Recuperación para condiciones de error en los ASF" en la página 347 y "Códigos de razón y de información de retorno en ASF" en la página 347.

Recuperación para condiciones de error en los ASF
Si un determinado ConnectionConsumer experimenta un problema grave, se detiene la entrega de mensajes a todos los ConnectionConsumers interesados en el mismo QLOCAL. En estos casos, se notifica cualquier ExceptionListener que se registre para la conexión afectada. Existen dos maneras en las que una aplicación se puede recuperar de estas condiciones de error.

Generalmente, ocurre un error grave de esta naturaleza si el ConnectionConsumer no puede volver a poner un mensaje en la cola de mensajes no entregados o si experimenta un error al leer mensajes de QLOCAL.

Debido a que cualquier ExceptionListener que esté registrado para la conexión afectada recibe una notificación, puede utilizarlos para identificar la causa del problema. En algunos casos, el administrador del sistema debe intervenir para resolver el problema.

Utilice una de las técnicas siguientes para efectuar la recuperación para estas condiciones de error:

- Invoque `close()` en todos los ConnectionConsumers afectados. La aplicación puede crear nuevos ConnectionConsumers sólo después de que se hayan cerrado todos los ConnectionConsumers afectados y se hayan resuelto todos los problemas del sistema.
- Invoque `stop()` en todas las conexiones afectadas. Después de que se hayan detenido todas las conexiones y se hayan resuelto los problemas del sistema, la aplicación puede `start()` sus conexiones correctamente.

Códigos de razón y de información de retorno en ASF
Utilice los códigos de razón y de información de retorno para determinar la causa de un error. En esta sección se proporcionan los códigos de razón habituales generados por el consumidor de conexión.

Para determinar la causa de un error, utilice la información siguiente:

- El código de información de retorno de todos los mensajes de informe
- El código de razón contenido en la MQDLH de todos los mensajes de la cola de mensajes no entregados

El consumidor de conexión genera los códigos de razón siguientes:

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Causa

El mensaje ha alcanzado al umbral de restitución definido para la QLOCAL, pero no se ha definido ninguna cola de retirada.

En las plataformas en las que no puede definir la cola de retirada, el mensaje ha alcanzado el umbral de restitución definido por JMS, que es 20.

Acción

Si no desea hacer esto, defina la cola de retirada para la QLOCAL correspondiente. Busque también la causa de las múltiples restituciones.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Causa

En la mensajería punto a punto, hay un mensaje que no coincide con ningún selector para la supervisión de la cola de ConnectionConsumers. Para mantener el rendimiento, el mensaje se reposiciona en la cola de mensajes no entregados.

Acción

Para evitar esta situación, asegúrese de que los ConnectionConsumers que utilizan la cola proporcionen un conjunto de selectores que trate todos los mensajes, o establezca QueueConnectionFactory para retener los mensajes.

De forma alternativa, determine el origen del mensaje.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Causa

JMS no puede interpretar el mensaje contenido en la cola.

Acción

Determine el origen del mensaje. Normalmente, JMS entrega los mensajes con un formato inesperado como BytesMessage o TextMessage. A veces, esto falla si el mensaje está muy mal formateado.

Otros códigos que aparecen en estos campos pueden ser debidos a que ha fallado un intento de recolocar el mensaje en una cola de retirada. En este caso, el código describe la razón por la que el reposicionamiento del mensaje en la cola ha fallado. Para diagnosticar la causa de estos errores, consulte API> Códigos de terminación y razón de la API.

Si el mensaje de informe no se puede colocar en la cola de respuesta (ReplyToQ), se coloca en la cola de mensajes no entregados. En esta situación, el campo de información de retorno de MQMD se completa tal como se describe en este tema. El campo de razón contenido en la MQDLH explica la razón por la que el mensaje de informe no se ha podido colocar en la cola de respuesta (ReplyToQ).

Función de una agrupación de sesiones del servidor en AFS

En este tema se resume la función de una agrupación de sesiones del servidor.

La Figura 45 en la página 349 resume los principios de las funciones de ServerSessionPool y ServerSession.

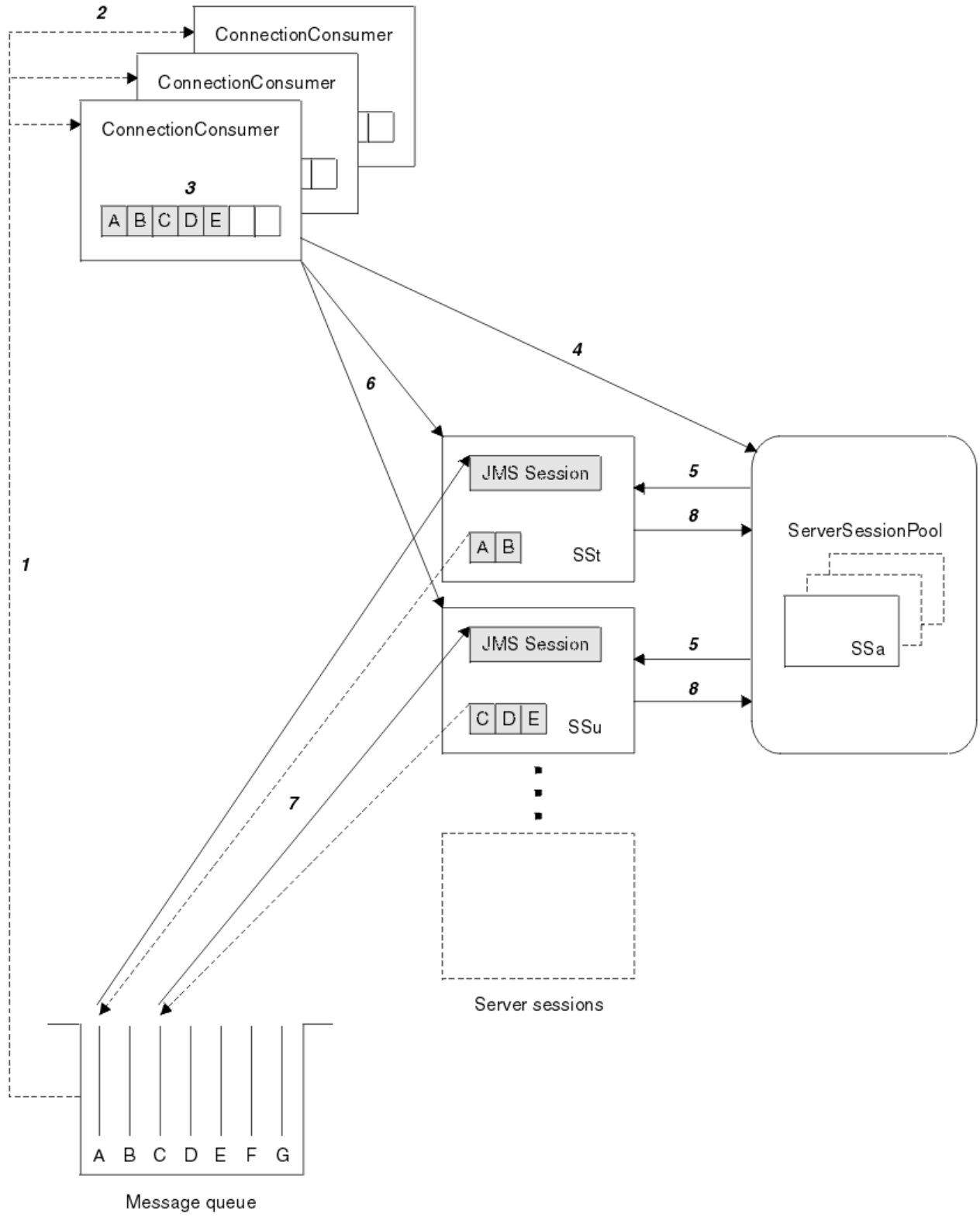


Figura 45. Funciones de ServerSessionPool y ServerSession

1. Los ConnectionConsumer obtienen referencias de mensaje de la cola.
2. Cada ConnectionConsumer selecciona referencias de mensaje específicas.
3. El almacenamiento intermedio de ConnectionConsumer contiene las referencias de mensaje seleccionadas.
4. ConnectionConsumer solicita una o más ServerSessions de la ServerSessionPool.

5. Se asignan ServerSessions a partir de la ServerSessionPool.
6. ConnectionConsumer asigna referencias de mensaje a las ServerSessions e inicia la ejecución de las hebras de ServerSession.
7. Cada ServerSession recupera sus mensajes referenciados de la cola. Pasa los mensajes al método onMessage desde el MessageListener que está asociado a la sesión de JMS.
8. Cuando finaliza su proceso, la ServerSession se devuelve a la agrupación.

Normalmente, un servidor de aplicaciones suministra las funciones de ServerSessionPool y ServerSession.

z/OS Utilización de IBM MQ classes for JMS en un servidor JVM de CICS Liberty

A partir de IBM MQ 9.1.0, los programas Java que ejecuten en un servidor de JVM CICS Liberty pueden usar las IBM MQ classes for JMS para acceder a IBM MQ.

Debe utilizar una versión de IBM MQ 9.1.0o posterior del adaptador de recursos de IBM MQ . Puede obtener el adaptador de recursos de Fix Central (consulte “Instalación del adaptador de recursos en Liberty” en la página 458).

Hay dos variantes de JVM de perfil Liberty disponibles en CICS 5.3 y posteriores, los posibles tipos de conexión con IBM MQ tienen las siguientes limitaciones:

CICS Liberty Estándar

- El adaptador de recursos IBM MQ se puede conectar a cualquier versión en servicio de IBM MQ en modalidad CLIENT
- El adaptador de recursos IBM MQ se puede conectar a cualquier versión en servicio de IBM MQ for z/OS en modalidad BINDINGS cuando no hay ninguna conexión CICS (definición de recurso CICS MQCONN activa) con el mismo gestor de colas de la misma región CICS.

CICS Liberty Integrado

- El adaptador de recursos IBM MQ se puede conectar a cualquier versión en servicio de IBM MQ en modalidad CLIENT.
- La conexión en modo BINDINGS no está soportada.

Para obtener detalles sobre la instalación y configuración del sistema, consulte Using IBM MQ classes for JMS in a Liberty JVM server en la documentación de CICS.

Utilización de IBM MQ classes for JMS/ Jakarta Messaging en IMS

El soporte de mensajería basado en estándares dentro de un entorno IMS se proporciona mediante el uso de IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Compruebe los requisitos del sistema para el sistema IMS que utiliza su empresa. Consulte IMS 15.2 para obtener más información.

Este conjunto de temas describe cómo configurar las IBM MQ classes for JMS en un entorno IMS y las restricciones de la API que se aplican cuando se utiliza la interfaz clásica de JMS 1.1 y la interfaz simplificada de JMS 2.0. Para obtener una lista de información específica de las API, consulte la sección “Restricciones del API JMS” en la página 355.

Nota: Se aplican restricciones similares a las interfaces específicas del dominio, JMS 1.0.2, pero no se describen detalladamente en esta sección.

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

Regiones dependientes de IMS soportadas

Se da soporte a los tipos de regiones dependientes siguientes:

- MPR
- BMP
- IFP
- Máquinas virtuales JMP 31 y 64 bits Java (JVM)
- JVM de 31 y 64 bits de JBP

A menos que se mencione específicamente en los temas siguientes, IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging se comportan igual en todos los tipos de región.

Java Virtual Machines soportadas

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging requieren IBM Runtime Environment, Java Technology Edition 8. IBM Semeru Runtime Certified Edition para z/OS, Versión 11 no está soportado.

Otras restricciones

Se aplican las restricciones siguientes cuando se utilizan las IBM MQ classes for JMS en un entorno IMS:

- No se da soporte a conexiones en modo cliente.
- Las conexiones sólo están soportadas en los gestores de colas de IBM MQ 8.0 utilizando el IBM MQ proveedor de mensajería `Normal`, modalidad.

  El atributo **PROVIDERVERSION** de la fábrica de conexiones no debe estar especificado o debe ser un valor superior o igual a siete.
- No está soportado el uso de cualquier fábrica de conexiones XA, por ejemplo, no está soportado `com.ibm.mq.jms.MQXAConnectionFactory`.

Tareas relacionadas

Definición de IBM MQ en IMS

Configuración del adaptador IMS para su uso con IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging utilizan el mismo adaptador IBM MQ-IMS que utilizan otros lenguajes de programación. Este adaptador utiliza el External Subsystem Attach Facility (ESAF) de IMS.

Antes de empezar

Antes de completar el procedimiento siguiente, debe configurar el adaptador de IMS para los gestores de cola apropiados, así como las regiones dependientes y control de IMS, según se describe en Configuración del adaptador IMS.

Atención: No es necesario que realice el paso que describe la creación de un apéndice dinámico, a menos que necesite el apéndice dinámico para otros fines.

Después de haber configurado el adaptador IMS , lleve a cabo el procedimiento siguiente.

Procedimiento

1. Actualice la variable LIBPATH en el miembro de su PROCLIB de IMS al que hace referencia el parámetro ENVIRON en el JCL de la región dependiente (por ejemplo, DFSJVMEV) de modo que incluya las bibliotecas nativas de IBM MQ classes for JMS.

Es decir, el directorio zFS que contiene `libmqjims.so`. Por ejemplo, DFSJVMEV podría tener el aspecto siguiente, donde la última línea es el directorio que contiene las bibliotecas nativas IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging :

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Añada IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging a la vía de acceso de clases de la JVM, utilizada por la región dependiente de IMS , actualizando la opción `java.class.path` .

   Para ello, siga las instrucciones de DFSJVMMS member of the IMS PROCLIB data set.

   Por ejemplo, puede utilizar el código siguiente, donde la línea en negrita indica la actualización:

V 9.3.0 JM 3.0 V 9.3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   Nota: Aunque hay muchos archivos jar diferentes disponibles en el directorio que contiene IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging, sólo necesita `com.ibm.mq.allclient.jar` (JMS 2.0) o `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).
3. Detenga y reinicie las regiones dependientes de IMS que utilizarán IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging.

Qué hacer a continuación

Cree y configure fábricas de conexiones y destinos.

Existen tres enfoques posibles para crear una instancia de las implementaciones de IBM MQ de las fábricas de conexiones y destinos. Consulte “Creación y configuración de fábricas de conexiones y destinos” en la página 210 para obtener detalles.

Tenga en cuenta que estos tres enfoques son todos válidos en un entorno de IMS.

Conceptos relacionados

Configuración del adaptador IMS

Definición de IBM MQ en IMS

Comportamiento transaccional

Los mensajes enviados y recibidos por IBM MQ classes for JMS en un entorno IMS siempre están asociados con la unidad de trabajo (UOW) de IMS que está activa en la tarea actual.

Esa UOW solo se puede completar invocando los métodos de confirmación o retrotracción sobre una instancia del objeto `com.ibm.ims.dli.tm.Transaction`, o mediante la finalización normal de la tarea de IMS, en cuyo caso la UOW se confirma implícitamente. Si la tarea de IMS finaliza de forma anómala, la UOW se retrotrae.

Como resultado de esto, los valores de los argumentos **`transacted`** y **`acknowledgeMode`** se pasan por alto al llamar a cualquiera de los métodos `Connection.createSession` o `ConnectionFactory.createContext`. Además, no se da soporte a los métodos siguientes. La llamada a cualquiera de los métodos siguientes da como resultado una `IllegalStateException` en el caso de sesión.

- `javax.jms.Session.commit()`

- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

y una excepción `IllegalStateRuntimeSession` en el caso de contexto de JMS:

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

Hay una excepción a este comportamiento. Si se crea una sesión o un contexto de JMS utilizando uno de los mecanismos siguientes:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

entonces el comportamiento de esa sesión o contexto de JMS es el siguiente:

- Cualquier mensaje enviado, se envía fuera de la UOW de IMS. Es decir, estarán disponibles en el destino inmediatamente, o cuando se haya completado el intervalo de retardo de entrega proporcionado.
- Los mensajes no persistentes se recibirán fuera de la UOW de IMS, a menos que se haya especificado la propiedad SYNCPOINTALLGETS en la fábrica de conexiones que ha creado la sesión o el contexto de JMS.
- Los mensajes persistentes siempre se recibirán dentro de la UOW de IMS.

Esto puede ser útil si, por ejemplo, desea escribir un mensaje de auditoría en una cola incluso si la UOW se retrotrae.

Implicaciones de los puntos de sincronismo de IMS

Las IBM MQ classes for JMS se basan en el soporte del adaptador de IBM MQ existente que utiliza ESAF. Esto significa que se aplica el comportamiento descrito cuando se produce un punto de sincronismo, incluido el cierre por parte del adaptador IMS de todos los manejadores abiertos.

Consulte “Puntos de sincronismo en aplicaciones IMS” en la página 73 para obtener más información.

Para ilustrar este punto, tenga en cuenta el código siguiente que se ejecuta en un entorno JMP. La segunda llamada `mp.send()` genera una `JMSException`, ya que el código `messageQueue.getUnique(inputMessage)` genera el cierre de todas las conexiones de IBM MQ y descriptores de objetos que estaban abiertos.

Se observa un comportamiento similar si la llamada `getUnique()` se sustituye por `Transaction.commit()`, pero no si se ha utilizado `Transaction.rollback()`.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

El código correcto que se ha de utilizar en este escenario es el siguiente. En este caso, se cierra la conexión con IBM MQ antes de la llamada `getUnique()`. Se vuelven a crear la conexión y la sesión para poder enviar otro mensaje.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

Consideraciones sobre el uso del adaptador IMS

Hay que tener presentes las siguientes restricciones. Solo se puede tener un descriptor de conexión por cada gestor de colas. Hay implicaciones en la interacción con IBM MQ cuando se utiliza código JMS y también código nativo. Existen limitaciones a la autenticación y autorización de conexiones.

Un descriptor de conexión por cada gestor de colas

Solo está permitido un descriptor de conexión a la vez en un gestor de colas específico en las regiones dependientes de IMS. Cualquier intento posterior de conectarse con el mismo gestor de colas reutilizará el descriptor de contexto existente.

Aunque este comportamiento no debería ocasionar ningún problema a una aplicación que solo use las IBM MQ classes for JMS, sí puede provocar problemas en aplicaciones que interactúen con IBM MQ cuando se usen tanto las IBM MQ classes for JMS y la MQI en código nativo escrito en lenguajes como, por ejemplo, COBOL o C.

Implicaciones de la interacción con IBM MQ cuando se utiliza tanto código JMS como código nativo

Se pueden producir problemas al intercalar el código Java y el código nativo que utilizan la funcionalidad IBM MQ y cuando la conexión con IBM MQ no se cierra antes de salir del código nativo o Java.

Por ejemplo, en el pseudocódigo siguiente, un descriptor de conexión a un gestor de colas se establece originalmente en el código Java utilizando IBM MQ classes for JMS. El descriptor de contexto se reutiliza en código COBOL y se invalida mediante una llamada a MQDISC.

La próxima vez que las IBM MQ classes for JMS utilicen el descriptor de conexión, se generará una `JMSException` con el código de razón MQRC_HCONN_ERROR.

```
COBOL code running in message processing region
```

```
Use the Java Native Interface (JNI) to call Java code
   Create MQ connection and session - this creates an MQ connection handle
   Send message to MQ queue
   Store connection and session in static variable
   Return to COBOL code

MQCONN - picks up MQ connection handle established in Java code
MQDISC - invalidates connection handle

Use the Java Native Interface (JNI) to call Java code
   Get session from static variable
   Create a message consumer - fails as connection handle invalidated
```

Hay otros patrones de uso similares que pueden resultar en un MQRC_HCONN_ERROR.

Aunque es posible compartir manejadores de conexión IBM MQ entre el código nativo y el código Java (por ejemplo, el ejemplo anterior funcionaría si no hubiera habido una llamada MQDISC) en general, se recomienda cerrar los manejadores de conexión antes de cambiar de Java a código nativo, o al revés.

Autenticación y autorización de conexiones

a especificación JMS permite que se especifique un nombre de usuario y una contraseña para la autenticación y la autorización cuando se crea una conexión o un objeto de contexto JMS.

Esto no está soportado en un entorno IMS. Al intentar crear una conexión al especificar un nombre de usuario y una contraseña, se genera un `JMS Exception` . Si se intenta crear un contexto JMS, mientras se especifica un nombre de usuario y una contraseña , se lanza una excepción de `JMSRuntimeException`.

En su lugar, se deben utilizar los mecanismos existentes para la autenticación y autorización cuando se conecta a IBM MQ desde un entorno IMS.

Para obtener más información, consulte Configurar la seguridad en z/OS. En particular, consulte ID de usuario para la comprobación de seguridad, que describe los ID de usuario que se pueden utilizar.

Tareas relacionadas

Configuración de la seguridad en z/OS

Restricciones del API JMS

Desde una perspectiva de especificación de JMS , los IBM MQ classes for JMS tratan IMS como un servidor de aplicaciones compatible con Java EE o Jakarta EE , que siempre tiene una transacción JTA en curso.

Por ejemplo, nunca se puede invocar `javax.jms.Session.commit()` en IMS, porque la especificación JMS establece que dicho método no se puede invocar en un EJB JEE ni en un contenedor web mientras haya una transacción JTA en curso.

Esto da lugar a las restricciones siguientes en la API JMS, además de las descritas en “Comportamiento transaccional” en la página 352.

Restricciones de la API clásica

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` siempre emite una `JMSException`.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` siempre emite una `JMSException`.
- Las tres variantes de `javax.jms.Connection.createSession` siempre generan una `JMSException` si la conexión ya tiene una sesión activa existente.
- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` siempre emite una `JMSException`.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` siempre emite una `JMSException`.
- `javax.jms.Connection.setClientID()` siempre emite una `JMSException`.

- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` siempre emite una `JMSException`.
- `javax.jms.Connection.stop()` siempre emite una `JMSException`.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` siempre emite una `JMSException`.
- `javax.jms.MessageConsumer.getMessageListener()` siempre emite una `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` siempre emite una `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` siempre emite una `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` siempre emite una `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener )` siempre emite una `JMSException`.
- `javax.jms.Session.run()` siempre emite una `JMSRuntimeException`.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` siempre emite una `JMSException`.
- `javax.jms.Session.getMessageListener()` siempre emite una `JMSException`.

Restricciones del API simplificada

- `javax.jms.JMSContext.createContext(int)` siempre emite una `JMSRuntimeException`.
- `javax.jms.JMSContext.setClientID(String)` siempre emite una `JMSRuntimeException`.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` siempre emite una `JMSRuntimeException`.
- `javax.jms.JMSContext.stop()` siempre emite una `JMSRuntimeException`.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` siempre emite una `JMSRuntimeException`.

Utilización de IBM MQ classes for Java

Utilice IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

Nota:

Stabilized IBM no hará mejoras adicionales en IBM MQ classes for Java y sus funciones se estabilizarán en el nivel suministrado en IBM MQ 8.0. Las aplicaciones existentes que utilizan IBM MQ classes for Java siguen recibiendo soporte completo, pero no se añadirán nuevas características y se rechazarán las solicitudes de mejoras. Soporte completo significa que los defectos se solucionarán junto con los cambios necesarios por los cambios en los requisitos del sistema IBM MQ.

IBM MQ classes for Java no se admite en IMS.

IBM MQ classes for Java no se admite en WebSphere Liberty. No se deben utilizar con la característica de mensajería de IBM MQ Liberty o con el soporte de JCA genérico. Para obtener más información, consulte Utilización de interfaces Java de WebSphere MQ en entornos J2EE/JEE.

IBM MQ classes for Java es una de las tres API alternativas que las aplicaciones Java pueden utilizar para acceder a los recursos de IBM MQ . Las otras API son:

- V 9.3.0 JM 3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

Para obtener más información, consulte “Acceso a IBM MQ desde Java -Opción de API” en la página 89.

A partir de IBM MQ 9.3, los IBM MQ classes for Java se crean con Java 8. El entorno de ejecución de Java 8 permite ejecutar versiones de archivos de clases anteriores.

IBM MQ classes for Java encapsula la interfaz de cola de mensajes (MQI), la API nativa de IBM MQ y utiliza un modelo de objeto similar a las interfaces C++ y .NET con IBM MQ.

Existen opciones programables que permiten que IBM MQ classes for Java se conecte a IBM MQ en una de las dos maneras siguientes:

- En la modalidad de cliente como IBM MQ MQI client utilizando el protocolo de control de transmisiones/protocolo Internet (TCP/IP)
- En la modalidad de enlaces, conectándose directamente a IBM MQ utilizando la interfaz nativa Java (JNI)

Nota: IBM MQ classes for Java no da soporte a la reconexión automática del cliente.

Conexión en modalidad de cliente

Una aplicación de IBM MQ classes for Java puede conectar con cualquier gestor de colas soportado utilizando la modalidad de cliente.

Para conectar con un gestor de colas en la modalidad de cliente, una aplicación de IBM MQ classes for Java se puede ejecutar en el mismo sistema en el que se ejecuta el gestor de colas, o en un sistema distinto. En cada caso, IBM MQ classes for Java se conecta con el gestor de colas a través de TCP/IP.

Para obtener más información sobre cómo escribir aplicaciones para utilizar conexiones en la modalidad de cliente, consulte “Modalidades de conexión de IBM MQ classes for Java” en la página 383.

Conexión de modalidad de enlaces

Cuando se utiliza en la modalidad de enlaces, IBM MQ classes for Java utiliza la interfaz nativa de Java (JNI) para llamar directamente en la API del gestor de colas existente, en lugar de comunicarse a través de una red. En la mayoría de los entornos, la conexión en la modalidad de enlaces proporciona un mejor rendimiento para las aplicaciones de IBM MQ classes for Java que la conexión en la modalidad de cliente al evitar el coste de la comunicación TCP/IP.

Las aplicaciones que utilizan IBM MQ classes for Java para conectar en la modalidad de enlaces se deben ejecutar en el mismo sistema que el gestor de colas al que se están conectando.

El entorno de ejecución de Java, que se está utilizando para ejecutar la aplicación IBM MQ classes for Java, se debe configurar para cargar las bibliotecas IBM MQ classes for Java; consulte “IBM MQ classes for Java bibliotecas” en la página 367 para obtener más información.

Para obtener más información sobre cómo escribir aplicaciones para utilizar conexiones en la modalidad de enlaces, consulte “Modalidades de conexión de IBM MQ classes for Java” en la página 383.

Conceptos relacionados

Interfaces de lenguaje de IBM MQ Java

“Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85

IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

Tareas relacionadas

Rastreo de aplicaciones de IBM MQ classes for Java

Resolución de problemas de Java y JMS

¿Por qué debo utilizar IBM MQ classes for Java?

Una aplicaciónJava puede utilizar IBM MQ classes for Java o IBM MQ classes for JMS para acceder a recursos IBM MQ.

Nota: V 9.3.0 V 9.3.0 Aunque las aplicaciones existentes que utilizan IBM MQ classes for Java siguen estando totalmente soportadas, las nuevas aplicaciones deben utilizar IBM MQ classes for Jakarta Messaging. Las características que se han añadido recientemente a IBM MQ, como el consumo asíncrono y la reconexión automática, no están disponibles en IBM MQ classes for Java, pero están disponibles en IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging. Para obtener más información, consulte "¿Por qué debo utilizar IBM MQ classes for JMS?" en la página 87 y "¿Por qué debo utilizar IBM MQ classes for Jakarta Messaging?" en la página 86.

Nota:

Stabilized IBM no hará mejoras adicionales en IBM MQ classes for Java y sus funciones se estabilizarán en el nivel suministrado en IBM MQ 8.0. Las aplicaciones existentes que utilizan IBM MQ classes for Java siguen recibiendo soporte completo, pero no se añadirán nuevas características y se rechazarán las solicitudes de mejoras. Soporte completo significa que los defectos se solucionarán junto con los cambios necesarios por los cambios en los requisitos del sistema IBM MQ.

IBM MQ classes for Java no se admite en IMS.

IBM MQ classes for Java no se admite en WebSphere Liberty. No se deben utilizar con la característica de mensajería de IBM MQ Liberty o con el soporte de JCA genérico. Para obtener más información, consulte Utilización de interfaces Java de WebSphere MQ en entornos J2EE/JEE.

Conceptos relacionados

"Acceso a IBM MQ desde Java -Opción de API" en la página 89
IBM MQ proporciona tres interfaces de lenguaje Java .

Requisitos previos para IBM MQ classes for Java

Para utilizar IBM MQ classes for Java, son necesarios algunos otros productos de software.

Para obtener información sobre los requisitos previos de IBM MQ classes for Java, consulte la página web de Requisitos del sistema para IBM MQ.

Para desarrollar aplicaciones de IBM MQ classes for Java, necesita un Kit de desarrollo de Java (JDK). Encontrará detalles sobre los JDK soportados para cada sistema operativo en la información de Requisitos del sistema para IBM MQ.

Para ejecutar aplicaciones de IBM MQ classes for Java, necesita los componentes de software siguientes:

- Un gestor de colas de IBM MQ para las aplicaciones que se conectan a un gestor de colas
- Un entorno de ejecución de Java (JRE) para cada sistema en el que se ejecutan aplicaciones. Se proporciona un JRE adecuado con IBM MQ.
- IBM i Para IBM i, QShell, que es la opción 30 del sistema operativo
- z/OS Para z/OS, z/OS UNIX System Services (z/OS UNIX)

Si necesita conexiones TLS para utilizar módulos criptográficos que han sido certificados como conformes con FIPS 140-2, necesita el proveedor FIPS de IBM Java JSSE (IBMJSSEFIPS). Cada JDK y JRE de IBM de la Versión 1.4.2 o posterior contiene IBMJSSEFIPS.

Puede utilizar las direcciones de Internet Protocol versión 6 (IPv6) en las IBM MQ classes for Java aplicaciones si IPv6 está soportado por la máquina virtual Java (JVM) y la implementación TCP/IP en el sistema operativo.

Ejecución de aplicaciones de IBM MQ classes for Java en Java EE

Existen determinadas restricciones y consideraciones de diseño que se deben tener en cuenta antes de utilizar IBM MQ classes for Java en Java EE.

IBM MQ classes for Java tiene restricciones cuando se utiliza dentro de un entorno Java Platform, Enterprise Edition (Java EE). También hay consideraciones adicionales que se deben tener en cuenta al

diseñar, implementar y gestionar una aplicación de IBM MQ classes for Java que se ejecuta dentro de un entorno Java EE. Estas limitaciones y consideraciones se destacan en los apartados siguientes.

Limitaciones de las transacciones JTA

El único gestor de transacciones soportado para aplicaciones que utilizan IBM MQ classes for Java es el propio IBM MQ. Aunque una aplicación bajo el control de JTA puede hacer uso de IBM MQ classes for Java, cualquier trabajo realizado a través de estas clases no está controlado por unidades de trabajo de JTA. En su lugar, forman unidades de trabajo locales separadas de las gestionadas por el servidor de aplicaciones a través de las interfaces de JTA. En particular, cualquier retrotracción de la transacción de JTA no da como resultado una retrotracción de los mensajes enviados o recibidos. Esta limitación se aplica a transacciones de aplicaciones o gestionadas por beans y a transacciones gestionadas por contenedores y a todos los contenedores de Java EE. Para realizar el trabajo de mensajería directamente con IBM MQ dentro de transacciones coordinadas por el servidor de aplicaciones, en su lugar se debe utilizar IBM MQ classes for JMS.

Creación de hebras

IBM MQ classes for Java crea hebras internamente para diversas operaciones. Por ejemplo, cuando se ejecuta en la modalidad BINDINGS para llamar directamente a un gestor de colas local, las llamadas se realizan en una 'hebra de trabajo' creada internamente por IBM MQ classes for Java. Se pueden crear otras hebras internamente, por ejemplo, para borrar las conexiones no utilizadas de una agrupación de conexiones o para eliminar suscripciones para las aplicaciones de publicación/suscripción terminadas.

Algunas aplicaciones de Java EE (por ejemplo, las que se ejecutan en contenedores EJB y Web) no deben crear nuevas hebras. En lugar de ello, todo el trabajo se debe realizar en las hebras de aplicación principales gestionadas por el servidor de aplicaciones. Cuando las aplicaciones utilizan IBM MQ classes for Java, el servidor de aplicaciones podría no distinguir entre el código de aplicación y el código de IBM MQ classes for Java, por lo que las hebras descritas anteriormente hacen que la aplicación no sea compatible con la especificación de contenedor. IBM MQ classes for JMS no infringe estas especificaciones de Java EE y por lo tanto se puede utilizar en su lugar.

Limitaciones de seguridad

Las políticas de seguridad implementadas por un servidor de aplicaciones pueden impedir determinadas operaciones que son realizadas por la API de IBM MQ classes for Java, tales como la creación y ejecución de nuevas hebras de control (tal como se describe en las secciones anteriores).

Por ejemplo, los servidores de aplicaciones se ejecutan normalmente con la seguridad de Java inhabilitada de forma predeterminada, y permiten que la seguridad se habilite a través de una configuración específica del servidor de aplicaciones (algunos servidores de aplicaciones también permiten una configuración más detallada de las políticas utilizadas dentro de la seguridad de Java). Cuando la seguridad de Java está habilitada, IBM MQ classes for Java podría romper las reglas de hebras de la política de seguridad de Java definidas para el servidor de aplicaciones, y la API podría no poder crear todas las hebras que necesita para que funcione. Para evitar problemas con la gestión de hebras, el uso de IBM MQ classes for Java no está soportado en los entornos en los que la seguridad de Java está habilitada.

Consideraciones sobre el aislamiento de las aplicaciones

Un beneficio previsto de ejecutar aplicaciones dentro de un entorno de Java EE es el aislamiento de aplicaciones. El diseño y la implementación de IBM MQ classes for Java son anteriores al entorno de Java EE . IBM MQ classes for Java se puede utilizar de una manera que no da soporte al concepto de aislamiento de aplicaciones. Los ejemplos específicos de consideraciones en esta área incluyen:

- El uso de valores estáticos (que abarcan todo el proceso JVM) en la clase MQEnvironment, tales como:
  - El ID de usuario y la contraseña que se utilizarán para identificación y autenticación de conexiones
  - El nombre de host, puerto y canal utilizados para las conexiones de cliente

- La configuración de TLS para conexiones de cliente seguras

  La modificación de cualquiera de las propiedades de MQEnvironment en provecho de una aplicación individual también afecta a otras aplicaciones que utilizan las mismas propiedades. Cuando se procesa en un entorno de varias aplicaciones, tal como Java EE, cada aplicación debe utilizar su propia configuración diferenciada mediante la creación de objetos MQQueueManager con un conjunto específico de propiedades, en lugar de usar de forma predeterminada las propiedades configuradas en la clase MQEnvironment de todo el proceso.

- La clase MQEnvironment aporta varios métodos estáticos que actúan globalmente en todas las aplicaciones que utilizan IBM MQ classes for Java dentro del mismo proceso de JVM, y no hay forma de alterar temporalmente este comportamiento para aplicaciones determinadas. Los ejemplos incluyen:
  - configurar propiedades de TLS, tales como la ubicación del almacén de claves
  - configurar salidas de canal de cliente
  - habilitar o inhabilitar el rastreo de diagnóstico
  - gestionar la agrupación de conexiones predeterminada que se utiliza para optimizar el uso de conexiones con gestores de colas

  La invocación de esos métodos afecta a todas las aplicaciones que se ejecutan en el mismo entorno de Java EE.

- La agrupación de conexiones está habilitada para optimizar el proceso de crear varias conexiones en el mismo gestor de colas. El gestor de agrupaciones de conexiones predeterminado abarca todo el proceso y se comparte entre varias aplicaciones. Los cambios en la configuración de la agrupación de conexiones, tales como la sustitución del gestor de conexiones predeterminado para una aplicación utilizando el método MQEnvironment.setDefaultConnectionManager (), afectan, por lo tanto, a otras aplicaciones que se ejecutan en el mismo servidor de aplicaciones de Java EE.
- TLS se configura para las aplicaciones que utilizan IBM MQ classes for Java mediante las propiedades de la clase MQEnvironment y del objeto MQQueueManager. No está integrado con la configuración de seguridad gestionada del propio servidor de aplicaciones. Debe asegurarse de que configura IBM MQ classes for Java debidamente para proporcionar el nivel de seguridad necesario y no utilizar la configuración del servidor de aplicaciones.

Restricciones de la modalidad de enlaces

IBM MQ y WebSphere Application Server se pueden instalar en la misma máquina de modo que las versiones principales del gestor de colas y del adaptador de recursos de IBM MQ que se proporcionan en WebSphere Application Server sean diferentes.

Si las versiones principales del gestor de colas y del adaptador de recursos son diferentes, no se pueden utilizar conexiones de enlaces. Las conexiones de WebSphere Application Server con el gestor de colas establecidas mediante el adaptador de recursos deben utilizar conexiones de tipo cliente. Se pueden utilizar conexiones de enlaces si las versiones son iguales.

Conversiones de cadenas de caracteres en IBM MQ classes for Java

Los IBM MQ classes for Java utilizan CharsetEncoders y CharsetDecoders directamente para la conversión de series de caracteres. El comportamiento predeterminado para la conversión de series de caracteres se puede configurar con dos propiedades del sistema. El manejo de mensajes que contienen caracteres no correlacionables se puede configurar a través de `com.ibm.mq.MQMD`.

Antes de IBM MQ 8.0, las conversiones de series en IBM MQ classes for Java se realizaban llamando a los métodos `java.nio.charset.Charset.decode(ByteBuffer)` y `Charset.encode(CharBuffer)`.

La utilización de cualquiera de estos métodos da lugar a una sustitución predeterminada (REPLACE) de datos malformados o no traducibles. Este comportamiento puede ocultar errores en las aplicaciones y dar lugar a caracteres inesperados, por ejemplo ?, en los datos traducidos.

A partir de IBM MQ 8.0, para detectar dichos problemas de forma más rápida y eficaz, IBM MQ classes for Java utilizan CharsetEncoders y CharsetDecoders directamente y configuran explícitamente el manejo de

los datos mal formados y no trducibles. El comportamiento predeterminado es REPORT estos problemas emitiendo un MQExceptionadecuado.

Configuración

La conversión de UTF-16 (la representación de caracteres utilizada en Java) z un juego de caracteres nativos como, UTF-8, se denomina *codificación*, mientras que la conversión en dirección opuesta se denomina *descodificación*.

La decodificación adopta el comportamiento predeterminado para CharsetDecoders, notificando errores emitiendo una excepción.

Se usa un parámetro de configuración para especificar una java.nio.charset.CodingErrorAction que controle el manejo de errores en la codificación y en la descodificación. Se usa otro parámetro de configuración para controlar el byte, o los bytes, de sustitución al codificar. Se usará la cadena Java de sustitución predeterminada en las operaciones de decodificación.

Configuración del manejo de datos no traducibles en IBM MQ classes for Java

A partir de IBM MQ 8.0, com.ibm.mq.MQMD incluye los dos campos siguientes:

byte[] unMappableReplacement
Secuencia de bytes que se escribe en una cadena codificada si no se puede traducir un carácter de entrada y se ha especificado REPLACE.

Valor predeterminado: "?".getBytes()
La cadena Java de sustitución predeterminada se usa en las operaciones de decodificación.

java.nio.charset.CodingErrorAction unmappableAction
Especifica la acción que hay que realizar con los datos no traducibles en codificación y decodificación:

Valor predeterminado: CodingErrorAction.REPORT;

Propiedades del sistema para establecer valores predeterminados del sistema

A partir de IBM MQ 8.0, las dos propiedades del sistema Java siguientes están disponibles para configurar el comportamiento predeterminado con respecto a la conversión de series de caracteres.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Especifica la acción que hay que realizar con los datos no traducibles en codificación y decodificación. El valor puede ser REPORT, REPLACE o IGNORE.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Establece u obtiene los bytes de sustitución que se aplican cuando un carácter no se puede correlacionar en una operación de codificación. La serie de sustitución Java predeterminada se utiliza en operaciones de descodificación.

Para evitar confusiones entre las representaciones de caracteres y de bytes nativos de Java, debe especificar com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement como un número decimal que representa el byte de sustitución en el juego de caracteres nativos.

Por ejemplo, el valor decimal de ?, como byte nativo, es 63 si el juego de caracteres nativo está basado en ASCII como, por ejemplo, ISO-8859-1, mientras que es 111 si el juego de caracteres nativo es EBCDIC.

Nota: Tenga en cuenta que si un objeto MQMD o MQMessage tiene establecidos los campos **unmappableAction** o **unMappableReplacement** , los valores de estos campos tienen prioridad sobre las propiedades del sistema Java . Esto permite que los valores especificados por las propiedades del sistema Java se alteren temporalmente para cada mensaje si es necesario.

Conceptos relacionados

“Conversiones de cadenas de caracteres en IBM MQ classes for JMS” en la página 143
Los IBM MQ classes for JMS utilizan CharsetEncoders y CharsetDecoders directamente para la conversión de series de caracteres. El comportamiento predeterminado para la conversión de series de caracteres se puede configurar con dos propiedades del sistema. El manejo de mensajes que contienen caracteres

no correlacionables se puede configurar mediante propiedades de mensaje para establecer la acción UnmappableCharactery los bytes de sustitución.

Instalación y configuración de IBM MQ classes for Java

Esta sección describe los directorios y archivos que se crean cuando se instala IBM MQ classes for Java y le indica cómo configurar IBM MQ classes for Java después de la instalación.

¿Qué se instala para IBM MQ classes for Java?

La versión más reciente de IBM MQ classes for Java se instala con IBM MQ. Puede ser necesario alterar temporalmente las opciones predeterminadas de la instalación para asegurarse de que esto se realice.

Para obtener más información sobre la instalación de IBM MQ, consulte:

- Multi Instalación de IBM MQ
- z/OS Instalación del producto IBM MQ for z/OS

IBM MQ classes for Java están contenidos en los archivos JAR (Java Archive), `com.ibm.mq.jar` y `com.ibm.mq.jmqi.jar`.

El soporte para cabeceras de mensaje estándar, tal como Programmable Command Format (PCF), está contenido en el archivo JAR `com.ibm.mq.headers.jar`.

El soporte para PCF (Programmable Command Format) está contenido en el archivo JAR `com.ibm.mq.pcf.jar`.

Nota: No es recomendable utilizar IBM MQ classes for Java dentro de un servidor de aplicaciones. Para obtener información sobre las restricciones que se aplican cuando se ejecuta en este entorno, consulte “Ejecución de aplicaciones de IBM MQ classes for Java en Java EE” en la página 358. Para obtener más información, consulte Utilización de interfaces WebSphere MQ Java en entornos J2EE/JEE.

Importante: Aparte de los archivos JAR reubicables descritos en “IBM MQ classes for Java Archivos JAR reubicables” en la página 362, no se da soporte a la copia de los archivos JAR de IBM MQ classes for Java o bibliotecas nativas en otras máquinas, o en una ubicación diferente en una máquina en la que se ha instalado IBM MQ classes for Java .

IBM MQ classes for Java Archivos JAR reubicables

Los archivos JAR reubicables se pueden mover a sistemas que necesitan ejecutar IBM MQ classes for Java.

Importante:

- Aparte de los archivos JAR reubicables descritos en Archivos JAR reubicables, no se da soporte a la copia de los archivos JAR de IBM MQ classes for Java o bibliotecas nativas en otras máquinas, o en una ubicación diferente en una máquina donde se ha instalado IBM MQ classes for Java .
- Para evitar conflictos de cargador de clases, no se recomienda empaquetar los archivos JAR reubicables dentro de varias aplicaciones dentro del mismo tiempo de ejecución de Java . En este escenario, considere la posibilidad de hacer que los archivos JAR reubicables de IBM MQ estén disponibles en la vía de acceso de clases del tiempo de ejecución de Java .
- No incluya los archivos JAR reubicables dentro de las aplicaciones desplegadas en los servidores de aplicaciones Java EE , como por ejemplo WebSphere Application Server. En estos entornos, el adaptador de recursos de IBM MQ debe desplegarse y utilizarse en su lugar, ya que contiene el IBM MQ classes for Java. Tenga en cuenta que WebSphere Application Server incluye el adaptador de recursos de IBM MQ , por lo que no es necesario desplegarlo manualmente en este entorno. Además, los IBM MQ classes for Java no están soportados en WebSphere Liberty. Para obtener más información, consulte “Liberty y el adaptador de recursos de IBM MQ” en la página 454.
- Si está empaquetando los archivos JAR reubicables dentro de las aplicaciones, asegúrese de incluir todos los archivos JAR de requisito previo tal como se describe en Archivos JAR reubicables. También debe asegurarse de que tiene los procedimientos adecuados para actualizar los archivos JAR

empaquetados como parte del mantenimiento de la aplicación, para asegurarse de que el IBM MQ classes for Java permanece actual y los problemas conocidos se vuelven a mediar.

Archivos JAR reubicables

Dentro de una empresa, los archivos siguientes se pueden trasladar a sistemas que necesitan ejecutar aplicaciones de IBM MQ classes for Java:

- JMS 2.0 `com.ibm.mq.allclient.jar` "1" en la página 363
- V 9.3.0 JM 3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` "2" en la página 363
- V 9.3.3 Removed `com.ibm.mq.traceControl.jar`
- V 9.3.5 `bcpkix-jdk18on.jar` "3" en la página 363
- V 9.3.0 V 9.3.0 `bcpkix-jdk15to18.jar` "4" en la página 363
- V 9.3.5 `bcprov-jdk18on.jar` "3" en la página 363
- V 9.3.0 V 9.3.0 `bcprov-jdk15to18.jar` "4" en la página 363
- V 9.3.5 `bcutil-jdk18on.jar` "3" en la página 363
- V 9.3.0 V 9.3.0 `bcutil-jdk15to18.jar` "4" en la página 363
- V 9.3.3 `jackson-annotations.jar`
- V 9.3.3 `jackson-core.jar`
- V 9.3.3 `jackson-databind.jar`
- `org.json.jar`

Notas:

1. JMS 2.0 y JMS 1.1
2. Jakarta Messaging 3.0
3. Continuous Delivery desde IBM MQ 9.3.5
4. Long Term Support y Continuous Delivery antes de IBM MQ 9.3.5

El proveedor de seguridad de Bouncy Castle y los archivos JAR de soporte de CMS

Los archivos del proveedor de seguridad Bouncy Castle y los archivos JAR de soporte de CMS son necesarios. Para obtener más información, consulte Soporte para JRE que no son deIBM con AMS.

V 9.3.5 Para Continuous Delivery desde IBM MQ 9.3.5, son necesarios los siguientes archivos JAR:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS Para Long Term Support y Continuous Delivery antes de IBM MQ 9.3.5, son necesarios los siguientes archivos JAR:

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

org.json.jar

El archivo `org.json.jar` es necesario si la aplicación IBM MQ classes for Java utiliza una CCDT en formato JSON.

com.ibm.mq.allclient.jar y com.ibm.mq.jakarta.client.jar

Los archivos `com.ibm.mq.allclient.jar` y `com.ibm.mq.jakarta.client.jar` contienen las clases IBM MQ classes for JMS, IBM MQ classes for Javay PCF y Headers Classes. Si mueve estos archivos a una nueva ubicación, asegúrese de que realiza los pasos necesarios para mantener esta nueva ubicación con los nuevos fixpacks de IBM MQ . Además, asegúrese de que el uso de los archivos se da a conocer al soporte de IBM si está obteniendo un arreglo temporal.

Para determinar la versión del archivo `com.ibm.mq.allclient.jar` o el archivo `com.ibm.mq.jakarta.client.jar` , utilice el mandato siguiente:

V 9.3.0 JM 3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

El ejemplo siguiente muestra algunos datos de salida de este mandato:

```
C:\Archivos de programa\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Archivos de programa/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar, jackson-core.jar y jackson-databind.jar

V 9.3.3

Los tres archivos JAR Jackson son necesarios si la aplicación IBM MQ classes for Java crea conexiones TLS seguras con un gestor de colas.

Directorios de instalación para IBM MQ classes for Java
Los archivos y los ejemplos de IBM MQ classes for Java se instalan en diversas ubicaciones de acuerdo con la plataforma. La ubicación del entorno de ejecución Java (JRE) que se instala con IBM MQ también varía de acuerdo con la plataforma.

Directorios de instalación para archivos de IBM MQ classes for Java

La Tabla 49 en la página 365 muestra las ubicaciones donde están instalados los archivos de IBM MQ classes for Java.

Tabla 49. Directorios de instalación de IBM MQ classes for Java	
Plataforma	**Directorio**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib
Linux Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Directorios de instalación para ejemplos

Con IBM MQ se proporcionan algunas aplicaciones de ejemplo, tales como los Programas de verificación de la instalación (IVP). La Tabla 50 en la página 365 muestra dónde están instaladas las aplicaciones de ejemplo. Los programas de ejemplo de IBM MQ classes for Java residen en un subdirectorio denominado `wmqjava`. Los ejemplos de PCF se encuentran en un subdirectorio llamado `pcf`.

Tabla 50. Directorios de ejemplos	
Plataforma	**Directorio**
AIX AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i IBM i	/QIBM/ProdData/mqm/java/samples
Linux Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Directorios de instalación para JRE

Las IBM MQ classes for JMS necesitan un entorno de ejecución Java 7 (o posterior) Java Runtime Environment (JRE). Con IBM MQ se instala un JRE adecuado. La Tabla 51 en la página 365 muestra dónde está instalado este JRE. Para ejecutar programas Java como los ejemplos proporcionados, utilizando este JRE, invoque explícitamente *JRE_LOCATION*`/bin/java` o añada *JRE_LOCATION*`/bin` a la variable de entorno PATH (o equivalente) para la plataforma, donde *JRE_LOCATION* es el directorio proporcionado en Tabla 51 en la página 365.

Tabla 51. Directorios de JRE	
Plataforma	**Directorio**
AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i	/QIBM/ProdData/mqm/java/jre
Linux Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows Windows	*MQ_INSTALLATION_PATH*\java\jre

Tabla 51. Directorios de JRE (continuación)	
Plataforma	**Directorio**
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Variables de entorno aplicables a IBM MQ classes for Java
Si desea ejecutar aplicaciones IBM MQ classes for Java , su vía de acceso de clases debe incluir los directorios IBM MQ classes for Java y de ejemplos.

Para que las aplicaciones IBM MQ classes for Java se ejecuten, su vía de acceso de clases debe incluir el directorio IBM MQ classes for Java adecuado. Para ejecutar las aplicaciones de ejemplo, la classpath también debe incluir los directorios de ejemplo adecuados. Esta información se puede proporcionar en el mandato de invocación Java o en la variable de entorno **CLASSPATH** .

Importante: El establecimiento de la opción Java `-Xbootclasspath` para incluir IBM MQ classes for Java no está soportado.

La Tabla 52 en la página 366 muestra el valor de **CLASSPATH** adecuado para utilizar en cada plataforma para ejecutar aplicaciones IBM MQ classes for Java , incluidas las aplicaciones de ejemplo.

*Tabla 52. Valor **CLASSPATH** para ejecutar aplicaciones IBM MQ classes for Java , incluidas las aplicaciones de ejemplo IBM MQ classes for Java*	
Plataforma	**CLASSPATH establecer**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/pcf

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Si compila utilizando la opción `-Xlint` , es posible que vea un mensaje que le avisa de que `com.ibm.mq.ese.jar` no está presente. Puede pasar por alto este aviso. Este archivo solo está presente si ha instalado Advanced Message Security.

Los scripts proporcionados con IBM MQ classes for JMS utilizan las siguientes variables de entorno:

MQ_JAVA_DATA_PATH
Esta variable de entorno especifica el directorio para la salida de anotaciones y de rastreo.

MQ_JAVA_INSTALL_PATH
Esta variable de entorno especifica el directorio donde se instalan IBM MQ classes for Java, tal como se muestra en los directorios de instalación de IBM MQ classes for Java.

MQ_JAVA_LIB_PATH
Esta variable de entorno especifica el directorio en el que se almacenan las bibliotecas de IBM MQ classes for Java, tal como se muestra en Ubicación de las bibliotecas de IBM MQ classes for Java para cada plataforma. Algunos scripts que se proporcionan con IBM MQ classes for Java, tales como IVTRun, utilizan esta variable de entorno.

Windows En Windows, todas las variables de entorno se establecen automáticamente durante la instalación.

Linux AIX En un sistema AIX and Linux, puede utilizar el script `setjmsenv` (si utiliza una JVM de 32 bits) o `setjmsenv64` (si utiliza una JVM de 64 bits) para establecer las variables de entorno. Estos scripts están en el directorio *MQ_INSTALLATION_PATH*/java/bin .

IBM i En IBM i, la variable de entorno **QIBM_MULTI_THREADED** debe establecerse en Y. Puede ejecutar aplicaciones de varias hebras del mismo modo que ejecuta aplicaciones de hebra única. Para obtener más información, consulte Configuración de IBM MQ con Java y JMS.

IBM MQ classes for Java necesita el entorno de ejecución Java (JRE) de Java 7. Para obtener información sobre la ubicación de un JRE adecuado para la instalación con IBM MQ, consulte "Directorios de instalación para IBM MQ classes for Java" en la página 364.

IBM MQ classes for Java bibliotecas
La ubicación de las bibliotecas de IBM MQ classes for Java varía según la plataforma. Especifique esta ubicación cuando inicie una aplicación.

Para especificar la ubicación de las bibliotecas de interfaz nativa Java (JNI), inicie la aplicación utilizando un mandato **java** con el formato siguiente:

```
java -Djava.library.path= library_path application_name
```

donde *vía_acceso_bibliotecas* es la vía de acceso de IBM MQ classes for Java, que incluye las bibliotecas JNI. La Tabla 53 en la página 367 muestra la ubicación de las bibliotecas de IBM MQ classes for Java para cada plataforma. En esta tabla, *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que se ha instalado IBM MQ.

Tabla 53. La ubicación de las bibliotecas de IBM MQ classes for Java para cada plataforma

Plataforma	**Directorio que contiene las bibliotecas de IBM MQ classes for Java**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Linux Linux (plataforma x86)	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux (plataformas POWER, x86-64 y zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliotecas de 64 bits)

Tabla 53. La ubicación de las bibliotecas de IBM MQ classes for Java para cada plataforma (continuación)	
Plataforma	**Directorio que contiene las bibliotecas de IBM MQ classes for Java**
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (bibliotecas de 32 bits y 64 bits)

Nota:

1. Linux AIX En AIX o Linux (plataforma Power), utilice las bibliotecas de 32 bits o las bibliotecas de 64 bits. Utilice únicamente las bibliotecas de 64 bits si está ejecutando la aplicación en una máquina virtual Java (JVM) de 64 bits y en una plataforma de 64 bits. De lo contrario, utilice las bibliotecas de 32 bits.
2. Windows En Windows, puede utilizar la variable de entorno PATH para especificar la ubicación de las bibliotecas de IBM MQ classes for Java, en lugar de especificar su ubicación en el mandato **java**.
3. IBM i Para utilizar IBM MQ classes for Java en la modalidad de enlaces en IBM i, asegúrese de que la biblioteca QMQMJAVA esté en la lista de bibliotecas.
4. z/OS En z/OS, puede utilizar una máquina virtual Java (JVM) de 32 bits o de 64 bits. No tiene que especificar qué bibliotecas se van a utilizar; IBM MQ classes for Java puede determinar automáticamente qué bibliotecas JNI se deben cargar.

Conceptos relacionados

Utilización de IBM MQ classes for Java
Después de instalar IBM MQ classes for Java, puede configurar la instalación de modo que ejecute sus propias aplicaciones.

Soporte de OSGi con IBM MQ classes for Java
OSGi proporciona una infraestructura que da soporte al despliegue de aplicaciones como paquetes. Estos paquetes OSGi se proporcionan como parte de las IBM MQ classes for Java.

OSGi proporciona una infraestructura Java de finalidad general, segura y gestionada, que soporta el despliegue de aplicaciones con formato de paquetes. Los dispositivos compatibles con OSGi pueden descargar e instalar paquetes, y también eliminarlos cuando ya no se necesitan. La infraestructura gestiona la instalación y actualización de los paquetes de forma dinámica y escalable.

Las IBM MQ classes for Java incluyen los paquetes OSGi siguientes.

com.ibm.mq.osgi.java_*version_number*.jar
Los archivos JAR permiten que las aplicaciones utilicen las IBM MQ classes for Java.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, este archivo JAR permite a las aplicaciones utilizar IBM MQ classes for JMS y IBM MQ classes for Java, y también incluye el código para manejar mensajes PCF.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Para JMS 2.0, este archivo JAR permite a las aplicaciones utilizar IBM MQ classes for JMS y IBM MQ classes for Java, y también incluye el código para manejar mensajes PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, este archivo JAR proporciona los requisitos previos para `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Para JMS 2.0, este archivo JAR proporciona los requisitos previos para `com.ibm.mq.osgi.allclient_version_number.jar`.

donde *version_number* es el número de versión de IBM MQ que está instalado.

Los paquetes se instalan en el subdirectorio `java/lib/OSGi` de su instalación de IBM MQ o en la carpeta `java\lib\OSGi` en Windows.

A partir de IBM MQ 8.0, utilice los paquetes `com.ibm.mq.osgi.allclient_8.0.0.0.jar` y `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` para las aplicaciones nuevas. Si se utilizan estos paquetes se elimina la restricción que impide ejecutar las IBM MQ classes for JMS y las IBM MQ classes for Java en la misma infraestructura OSGi. No obstante, todas las otras restricciones se continúan aplicando. Para las versiones de IBM MQ anteriores a la IBM MQ 8.0, se aplica la restricción de utilizar las IBM MQ classes for JMS o las IBM MQ classes for Java.

También se instalan otros nueve paquetes en el subdirectorio `java/lib/OSGi` de su instalación de IBM MQ o en la carpeta `java\lib\OSGi` en Windows. Estos paquetes forman parte de las IBM MQ classes for JMS y no se deben cargar en un entorno de tiempo de ejecución de OSGi que tenga cargado el paquete de las IBM MQ classes for Java. Si se ha cargado el paquete OSGi de las IBM MQ classes for Java en un entorno de ejecución de OSGi que también tiene cargados los paquetes de las IBM MQ classes for JMS, se generan errores como los que se muestran en el ejemplo siguiente, cuando las aplicaciones utilizan el paquete de las IBM MQ classes for Java o cuando se ejecutan los paquetes de las IBM MQ classes for JMS:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

El paquete OSGi para las IBM MQ classes for Java se ha escrito en la especificación OSGi Release 4. NO funciona en un entorno OSGi Release 3.

Debe establecer correctamente la vía de acceso del sistema o de bibliotecas, de forma que el entorno de ejecución OSGi pueda encontrar los archivos DLL o las bibliotecas compartidas necesarias.

Si utiliza el paquete OSGi para las IBM MQ classes for Java, las clases de salida de canal escritas en Java no están soportadas debido a un problema inherente en la carga de clases en un entorno de múltiples cargadores de clases, como es el caso de OSGi. Un paquete de usuario puede reconocer el paquete de las IBM MQ classes for Java pero el paquete de las IBM MQ classes for Java no reconoce el paquete de usuario. Como resultado, el cargador de clases que se utiliza en un paquete de las IBM MQ classes for Java no puede cargar una clase de salida de canal si está en un paquete de usuario.

Para obtener más información sobre OSGi, consulte el sitio web de OSGi Alliance.

z/OS *Instalación de IBM MQ classes for Java en z/OS*

En z/OS, la STEPLIB utilizada en tiempo de ejecución debe contener las bibliotecas SCSQAUTH y SCSQANLE de IBM MQ.

A partir de z/OS UNIX System Services, puede añadir estas bibliotecas utilizando una línea en la `.profile` tal como se muestra en el ejemplo siguiente, sustituyendo `thlqual` por el calificador de conjunto de datos de alto nivel que ha elegido al instalar IBM MQ:

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

En otros entornos, por lo general, debe editar el JCL de arranque para incluir SCSQAUTH en la concatenación de STEPLIB:

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

El archivo de configuración de IBM MQ classes for Java

El archivo de configuración de IBM MQ classes for Java especifica las propiedades que se utilizan para configurar IBM MQ classes for Java.

Un archivo de configuración de IBM MQ classes for Java tiene el formato de un archivo de propiedades estándar de Java.

Se proporciona un archivo de configuración de ejemplo, `mqjava.config`, en el subdirectorio `bin` del directorio de instalación de IBM MQ classes for Java. Este archivo documenta todas las propiedades soportadas y sus valores predeterminados.

Nota: El archivo de configuración de ejemplo se sobrescriben cuando la instalación de IBM MQ se actualiza a un fixpack futuro. Por lo tanto, se recomienda que realice una copia del archivo de configuración de ejemplo para utilizarlo con las aplicaciones.

Puede elegir el nombre y la ubicación de un archivo de configuración de IBM MQ classes for Java. Al iniciar la aplicación, utilice un mandato **java** con el formato siguiente:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

En el mandato, *url_archivo_configuración* es un localizador uniforme de recursos (URL) que especifica el nombre y la ubicación del archivo de configuración de IBM MQ classes for Java. Los URL de los tipos siguientes están soportados: `http`, `file`, `ftp` y `jar`.

El ejemplo siguiente muestra un mandato **java**:

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

Este mandato identifica el archivo de configuración IBM MQ classes for Java como el archivo `D:\mydir\mqjava.config` en el sistema Windows local.

Cuando se inicia una aplicación, IBM MQ classes for Java lee el contenido del archivo de configuración y almacena las propiedades especificadas en un almacén de propiedades interno. Si el mandato **java** no identifica un archivo de configuración o si este no se puede encontrar, IBM MQ classes for Java utiliza los valores predeterminados para todas las propiedades. Si es necesario, puede alterar temporalmente cualquier atributo contenido en el archivo de configuración especificando el atributo como propiedad del sistema en el mandato **java**.

Un archivo de configuración de IBM MQ classes for Java se puede utilizar con cualquiera de los transportes soportados entre una aplicación y un gestor de colas o intermediario.

Alteración temporal de las propiedades especificadas en un archivo de configuración de IBM MQ MQI client

Un archivo de configuración de IBM MQ MQI client también puede especificar las propiedades que se utilizan para configurar IBM MQ classes for Java. Pero las propiedades que se especifican en un archivo de configuración de IBM MQ MQI client sólo son aplicables cuando una aplicación se conecta a un gestor de colas en la modalidad de cliente.

Si es necesario, puede alterar temporalmente cualquier atributo contenido en un archivo de configuración de IBM MQ MQI client especificando el atributo como propiedad en un archivo de configuración de IBM MQ classes for Java. Para alterar temporalmente un atributo en un archivo de configuración de IBM MQ MQI client, utilice una entrada con el formato siguiente en el archivo de configuración de IBM MQ classes for Java:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Las variables de la entrada tienen los significados siguientes:

stanza
El nombre de la stanza en el archivo de configuración de IBM MQ MQI client donde reside el atributo.

propName
El nombre del atributo tal como está especificado en el archivo de configuración de IBM MQ MQI client.

propValue
El valor de la propiedad que altera temporalmente el valor del atributo especificado en el archivo de configuración de IBM MQ MQI client.

Como alternativa, puede alterar temporalmente un atributo contenido en un archivo de configuración de IBM MQ MQI client especificando el atributo como propiedad del sistema en el mandato **java**. Utilice el formato anterior para especificar el atributo como propiedad del sistema.

Sólo los atributos siguientes contenidos en un archivo de configuración de IBM MQ MQI client son válidos para IBM MQ classes for Java. Si especifica o altera temporalmente otros atributos, dicha acción no tendrá afecto. En particular, observe que no se utilizan los atributos `ChannelDefinitionFile` y `ChannelDefinitionDirectory` contenidos en la Stanza CHANNELS del archivo de configuración de cliente. Consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for Java" en la página 387 para conocer detalles sobre cómo utilizar la tabla de definición de canal de cliente (CCDT) con IBM MQ classes for Java.

Tabla 54. Stanzas del archivo de configuración de cliente y los atributos que contienen

Stanza	**Atributo**
Stanza CHANNELS del archivo de configuración de cliente	Put1DefaultAlwaysSync
Stanza CHANNELS del archivo de configuración de cliente	PasswordProtection
Stanza ClientExitPath del archivo de configuración de cliente	Vía de acceso predeterminada de las salidas
Stanza ClientExitPath del archivo de configuración de cliente	ExitsDefaultPath64
Stanza ClientExitPath del archivo de configuración de cliente	JavaExitsClasspath
Stanza JMQI del archivo de configuración de cliente	useMQCSPauthentication
Stanza MessageBuffer del archivo de configuración de cliente	MaximumSize
Stanza MessageBuffer del archivo de configuración de cliente	PurgeTime
Stanza MessageBuffer del archivo de configuración de cliente	UpdatePercentage
Stanza TCP del archivo de configuración de cliente	ClntRcvBuffSize
Stanza TCP del archivo de configuración de cliente	ClntSndBuffSize
Stanza TCP del archivo de configuración de cliente	Connect_Timeout
Stanza TCP del archivo de configuración de cliente	KeepAlive

Para obtener más información sobre la configuración de IBM MQ MQI client , consulte el archivo de configuración de IBM MQ MQI client , `mqclient.ini`.

Tareas relacionadas

Rastreo de aplicaciones de IBM MQ Classes para Java

Utilización del rastreo de Java Standard Environment para configurar el rastreo de Java
Utilice la stanza Standard Environment Trace Settings de Java para configurar el recurso de rastreo de las IBM MQ classes for Java.

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputName*
traceOutputName es el directorio y el nombre de archivo al que se envía la salida de rastreo.

De forma predeterminada, la información de rastreo se escribe en un archivo de rastreo en el directorio de trabajo actual de la aplicación. El nombre del archivo de rastreo depende del entorno en el que se está ejecutando la aplicación:

- Desde IBM MQ 9.1.5 y IBM MQ 9.1.0 Fix Pack 5:
  - Si la aplicación ha cargado IBM MQ classes for Java desde el archivo JAR reubicable `com.ibm.mq.allclient.jar`, el rastreo se graba en un archivo denominado `mqjavaclient_%PID%.cl%u.trc`.
  - Si la aplicación ha cargado IBM MQ classes for Java desde el archivo JAR `com.ibm.mq.jar`, el rastreo se graba en un archivo denominado `mqjava_%PID%.cl%u.trc`.
- Desde IBM MQ 9.0.0 Fix Pack 2:
  - Si la aplicación ha cargado IBM MQ classes for Java desde el archivo JAR reubicable `com.ibm.mq.allclient.jar`, el rastreo se graba en un archivo denominado `mqjavaclient_%PID%.trc`.
  - Si la aplicación ha cargado IBM MQ classes for Java desde el archivo JAR `com.ibm.mq.jar`, el rastreo se graba en un archivo denominado `mqjava_%PID%.trc`.
- Para IBM MQ classes for Java para IBM MQ 9.0.0 Fix Pack 1 o anterior, el rastreo se escribe en un archivo llamado `mqjms_%PID%.trc`.

donde *%PID%* es el identificador de proceso de la aplicación que se está rastreando, y *%u* es un número único para diferenciar los archivos entre las hebras que ejecutan los classloaders de Java.

Si un ID de proceso no está disponible, se genera un número aleatorio con la letra `f`como prefijo. Para incluir el ID de proceso en un nombre de archivo que especifique, utilice la serie `%PID%`.

Si especifica otro directorio, éste debe existir, y debe tener permisos de escritura para el mismo. Si no tiene permisos de escritura, la salida de rastreo se escribe en `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList es una lista de los paquetes y clases que se rastrean, o los valores especiales `ALL` o `NONE`.

Separe los nombres de paquete y clase con un punto y coma, `;`. *includeList* toma como valor predeterminado `ALL` y rastrea todos los paquetes y clases de las IBM MQ classes for Java.

Nota: Puede incluir un paquete pero, a continuación, excluya los subpaquetes de dicho paquete. Por ejemplo, si incluye el paquete `a.b` y excluye `a.b.x`, el rastreo incluye todo el contenido de `a.b.y` y `a.b.z`, pero no el contenido de `a.b.x` o `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList es una lista de paquetes y clases que no se rastrean, o los valores especiales `ALL` o `NONE`.

Separe los nombres de paquete y clase con un punto y coma, `;`. *excludeList* toma como valor predeterminado `NONE` y, por lo tanto, no excluye del rastreo ningún paquete ni las IBM MQ classes for JMS del rastreo.

Nota: Puede excluir un paquete pero, a continuación, incluir subpaquetes de dicho paquete. Por ejemplo, si excluye el paquete `a.b` e incluye el paquete `a.b.x`, el rastreo incluye todo el contenido de `a.b.x` y de `a.b.x.1` pero no así lo de `a.b.y` o `a.b.z`.

Cualquier paquete o clase que se haya especificado, en el mismo nivel, ya que se incluye los incluidos y los excluidos.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes es el número máximo de bytes que se rastrean de cualquier matriz de bytes.

Si *maxArrayBytes* se establece en un entero positivo, limita el número de bytes de la matriz de bytes que se escriben en el archivo de rastreo. Trunca la matriz de bytes tras escribir *maxArrayBytes*. Definir *maxArrayBytes* reduce el tamaño del archivo de rastreo resultante y reduce el efecto de rastrear el rendimiento de la aplicación.

Un valor `0` para esta propiedad indica que no se envía el contenido de ninguna de las matrices de bytes al archivo de rastreo.

El valor predeterminado es `-1`, que elimina cualquier límite en el número de bytes en una matriz de bytes que se envían al archivo de rastreo.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes es el número máximo de bytes que se escriben en un archivo de salida de rastreo.

maxTraceBytes funciona con *traceCycles*. Si el número de bytes de rastreo escritos se acerca al límite, el archivo se cierra y se inicia un nuevo archivo de salida de rastreo.

Un valor `0` significa que un archivo de salida de rastreo tiene longitud cero. El valor predeterminado es `-1`, lo que significa que la cantidad de datos que se grabarán en el archivo de salida de rastreo es ilimitada.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles es el número de archivos de rastreo de salida que hay que recorrer.

Si el archivo de salida de rastreo actual alcanza el límite especificado por *maxTraceBytes*, el archivo se cierra. La salida de rastreo adicional se graba en el archivo de salida de rastreo siguiente de la secuencia. Cada archivo de salida de rastreo se distingue por un sufijo numérico que se añade al nombre de archivo. El archivo de salida de rastreo actual o más reciente es `mqjms.trc.0`, el siguiente archivo de salida de rastreo más reciente es `mqjms.trc.1`. Los archivos de rastreo más antiguos siguen el mismo patrón de numeración hasta el límite.

El valor predeterminado de *traceCycles* es 1. Si *traceCycles* es 1, cuando el archivo de salida de rastreo actual alcanza su tamaño máximo, el archivo se cierra y se suprime. Se inicia un nuevo archivo de salida de rastreo con el mismo nombre. Por consiguiente, únicamente existe un archivo de salida de rastreo simultáneamente.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters controla si los parámetros de método y valores de retorno se incluyen en el rastreo.

traceParameters es, de forma predeterminada, TRUE. Si *traceParameters* se establece en FALSE, sólo se rastrean las signaturas de método.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Existe una fase de inicialización de las IBM MQ classes for Java durante la cual se asignan los recursos. El recurso de rastreo principal se inicializa durante la fase de asignación de recursos.

Si *startup* se establece en TRUE, se utiliza el rastreo de inicio. La información de rastreo se produce inmediatamente e incluye la configuración de todos los componentes, incluido el propio recurso de rastreo. La información de rastreo inicial puede utilizarse para diagnosticar problemas de configuración. La información de rastreo de inicio siempre se graba en `System.err`.

startup es, de forma predeterminada, FALSE.

startup se comprueba antes de que finalice la inicialización. Por este motivo, especifique solo la propiedad en la línea de mandatos como una propiedad del sistema Java. No la especifique en el archivo de configuración de las IBM MQ classes for Java.

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Establezca *compressedTrace* en TRUE para comprimir la salida de rastreo.

El valor predeterminado de *compressedTrace* es FALSE.

Si *compressedTrace* se establece en TRUE, la salida de rastreo se comprime. El nombre del archivo de salida de rastreo tiene la extensión `.trz`. Si la compresión se establece en FALSE, que es el valor predeterminado, el archivo tiene la extensión `.trc` para indicar que no está comprimido. Sin embargo, si el nombre de archivo para la salida de rastreo se ha especificado en *traceOutputName*, se utiliza dicho nombre; no se aplica ningún sufijo al archivo.

La salida de rastreo comprimida es más pequeña que la no comprimida. Dado que significa menos E/S, puede escribirse con mayor rapidez que el archivo no comprimido. El rastreo comprimido afecta menos al rendimiento de las IBM MQ classes for Java que el rastreo no comprimido.

Si se ha establecido *maxTraceBytes* y *traceCycles*, se crean varios archivos de rastreo comprimidos en lugar de varios archivos planos.

Si las IBM MQ classes for Java finalizan de forma no controlada, es posible que el archivo de rastreo comprimido no sea válido. Por este motivo, solo se debe utilizar la compresión del rastreo cuando las IBM MQ classes for Java concluyen de modo controlado. Utilice la compresión de rastreo solo si los problemas que se están investigando no dan lugar a que la JVM se cierre de forma inesperada. No utilice la compresión de rastreo si diagnostica problemas que pueden dar lugar al cierre de `System.Halt()` o a una terminación no controlada de la JVM.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*

traceLevel especifica un nivel de filtrado distinto para el rastreo. Los niveles de rastreo definidos son los siguientes:

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

Cada nivel de rastreo incluye todos los niveles inferiores. Por ejemplo, si el nivel de rastreo se establece en TRACE_INFO, los puntos de rastreo con un nivel definido de TRACE_EXCEPTION, TRACE_WARNING o TRACE_INFO se escriben en el rastreo. Todos los demás puntos de rastreo se excluyen.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace controla si se utiliza el servicio de rastreo de cliente de IBM MQ classes for Java en un entorno WebSphere Application Server.

Si *standaloneTrace* está establecido en TRUE, se utilizan las propiedades de rastreo del cliente de IBM MQ classes for Java para determinar la configuración del rastreo.

Si *standaloneTrace* está establecido en FALSE, y el cliente de IBM MQ classes for Java se está ejecutando en un contenedor de WebSphere Application Server, se utiliza el servicio de rastreo de WebSphere Application Server. La información de rastreo que se genera depende de los valores de rastreo del servidor de aplicaciones.

El valor predeterminado de *standaloneTrace* es FALSE.

IBM MQ classes for Java y herramientas de gestión de software

Herramientas de gestión de software tales como Apache Maven se pueden usar con las IBM MQ classes for Java.

Muchas organizaciones de desarrollo de gran tamaño utilizan estas herramientas para gestionar de forma centralizada los repositorios de bibliotecas de terceros.

Las IBM MQ classes for Java constan de una serie de archivos JAR. Cuando se desarrollan aplicaciones de lenguaje Java utilizando esta API, es necesaria una instalación de IBM MQ Server, IBM MQ Client o IBM MQ Client SupportPac en la máquina en la que se está desarrollando la aplicación.

Si desea utilizar una herramienta de gestión de software y añade los archivos JAR que conforman IBM MQ classes for Java a un repositorio gestionado centralmente, se deben tener en cuenta los puntos siguientes:

- Hay que poner un repositorio o contenedor a disposición de los desarrolladores de la organización únicamente. No se permite ninguna distribución fuera de la organización.

- El repositorio debe contener un conjunto completo y coherente de archivos JAR de un único release o fixpack de IBM MQ.
- Usted es responsable de actualizar el repositorio con cualquier mantenimiento proporcionado el equipo de soporte de IBM.

Desde IBM MQ 8.0, el archivo JAR `com.ibm.mq.allclient.jar` se debe instalar en el repositorio.

A partir de IBM MQ 9.0, son necesarios los archivos JAR de soporte CMS y del proveedor de seguridad Bouncy Castle. Puede obtener información adicional consultando "IBM MQ classes for Java Archivos JAR reubicables" en la página 362 y Soporte de JRE que no son de IBM.

Configuración posterior a la instalación para aplicaciones de IBM MQ classes for Java

Después de instalar IBM MQ classes for Java, puede configurar la instalación de modo que ejecute sus propias aplicaciones.

Recuerde examinar el archivo readme del producto IBM MQ para obtener la información más reciente, o información más específica sobre el entorno. La versión más reciente del archivo readme del producto está disponible en la página web de IBM MQ, WebSphere MQ, y los archivos léame del producto MQSeries.

Antes de intentar ejecutar una aplicación de IBM MQ classes for Java en la modalidad de enlaces, asegúrese de que ha configurado IBM MQ tal como se describe en Configuración.

Configuración del gestor de colas para aceptar conexiones de cliente desde IBM MQ classes for Java

Para configurar el gestor de colas para que acepte las solicitudes de conexión entrantes de los clientes, defina y permita el uso de un canal de conexión de servidor e inicie un programa de escucha.

Consulte "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097 para obtener los detalles.

Ejecución de aplicaciones de IBM MQ classes for Java en el Java security manager

IBM MQ classes for Java puede ejecutarse con el Java security manager habilitado. Para ejecutar correctamente aplicaciones con el Java security manager habilitado, debe configurar la Java Virtual Machine (JVM) con un archivo de definiciones de política adecuado.

La forma más sencilla de crear un archivo de definición de políticas adecuado es cambiar el archivo de políticas que se proporciona con el Java runtime environment (JRE). En la mayoría de los sistemas, este archivo se almacena en `path lib/security/java.policy`, en relación con el directorio JRE. Puede editar los archivos de políticas utilizando su editor preferido o el programa policytool que se proporciona con el JRE.

Debe otorgar autorización al archivo `com.ibm.mq.jmqi.jar` para que pueda:

- Crear sockets (en la modalidad de cliente)
- Cargar la biblioteca nativa (en la modalidad de enlaces)
- Leer varias propiedades del entorno

La propiedad del sistema **`os.name`** debe estar disponible para IBM MQ classes for Java cuando se ejecuta con el Java security manager.

Si la aplicación Java utiliza el Java security manager, debe añadir el siguiente permiso al archivo `java.security.policy` que utiliza la aplicación; de lo contrario, se generarán excepciones en la aplicación:

```
permission java.lang.RuntimePermission "modifyThread";
```

Este RuntimePermission es necesario en el cliente como parte de la gestión de asignaciones y cierres de conversaciones multiplexadas a través de conexiones TCP con el gestor de colas.

Ejemplo de entrada de archivo de política

A continuación, se muestra un ejemplo de una entrada de archivo de política que permite a IBM MQ classes for Java ejecutarse correctamente con el gestor de seguridad predeterminado. Sustituya la serie *MQ_INSTALLATION_PATH* en este ejemplo por la ubicación donde se instala IBM MQ classes for Java en el sistema.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";
```

```
// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

Este ejemplo de un archivo de política permite que IBM MQ classes for Java funcione correctamente con el gestor de seguridad, pero es posible que todavía tenga que habilitar su propio código para que se ejecute correctamente para que funcionen las aplicaciones.

El código de ejemplo que se suministra con IBM MQ classes for Java no se ha habilitado específicamente para su uso con el gestor de seguridad; no obstante, las pruebas IVT se ejecutan con este archivo de políticas y el gestor de seguridad predeterminado en su lugar.

Importante:

El recurso de rastreo IBM MQ classes for Java requiere más permisos cuando realiza consultas adicionales de propiedades del sistema y también cuando realiza más operaciones del sistema de archivos.

En el directorio `samples/wmqjava` de la instalación de IBM MQ, se proporciona un archivo de política de seguridad de plantilla adecuado para que se ejecute bajo un gestor de seguridad con el rastreo habilitado como `example.security.policy`.

Para una instalación predeterminada, el archivo `example.security.policy` se encuentra en:

Windows
En `C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

Linux
En `/opt/mqm/samp/wmqjava/samples/example.security.policy`

Solaris
En `/opt/mqm/samp/wmqjava/samples/example.security.policy`

AIX
En `/usr/mqm/samp/wmqjava/samples/example.security.policy`

Ejecución de aplicaciones de IBM MQ classes for Java en CICS Transaction Server
Una aplicación de IBM MQ classes for Java se puede ejecutar como una transacción bajo CICS Transaction Server.

Para ejecutar una aplicación de IBM MQ classes for Java como una transacción bajo CICS Transaction Server para z/OS, siga estos pasos:

1. Defina la aplicación y la transacción para CICS utilizando la transacción CEDA proporcionada.
2. Asegúrese de que el adaptador IBM MQ CICS está instalado en el sistema CICS. z/OS (Consulte Utilización de IBM MQ con CICS para ver detalles.)
3. Asegúrese de que el entorno de JVM especificado en CICS incluya las entradas de CLASSPATH y LIBPATH adecuadas.
4. Inicie la transacción utilizando cualquiera de los procesos habituales.

Para obtener más información sobre cómo ejecutar transacciones CICS Java, consulte la documentación del sistema CICS.

Verificación de la instalación de IBM MQ classes for Java

Se proporciona un programa de verificación de instalación, MQIVP, con IBM MQ classes for Java. Puede utilizar este programa para probar todas las modalidades de conexión de IBM MQ classes for Java.

El programa le solicita que seleccione entre diversas opciones y otros datos para determinar la modalidad de conexión que desea verificar. Utilice el procedimiento siguiente para verificar la instalación:

1. Si va a ejecutar el programa en modalidad de cliente, configure el gestor de colas tal como se describe en "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097. La cola que se va a utilizar es SYSTEM.DEFAULT.LOCAL.QUEUE
2. Si el programa se ejecutará en la modalidad de cliente, consulte también "Utilización de IBM MQ classes for Java" en la página 356.

   Ejecute los demás pasos de este procedimiento en el sistema en el que va a ejecutar el programa.
3. Asegúrese de haber actualizado la variable de entorno CLASSPATH siguiendo las instrucciones indicadas en "Variables de entorno aplicables a IBM MQ classes for Java" en la página 366.
4. Cambie el directorio a *MQ_INSTALLATION_PATH*`/mqm/samp/wmqjava/samples`, donde *MQ_INSTALLATION_PATH* es la vía de acceso de la instalación de IBM MQ. En el indicador de mandatos, escriba:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   donde *vía_acceso_biblioteca* es la vía de acceso de las bibliotecas de IBM MQ classes for Java (consulte "IBM MQ classes for Java bibliotecas" en la página 367).

   En el mensaje de solicitud marcado (1):

   - Para utilizar una conexión TCP/IP, escriba un nombre de host del servidor IBM MQ.
   - Para utilizar la conexión nativa (modalidad de enlaces), deje el campo en blanco (no especifique un nombre).

   El programa realiza estas acciones:

   - 1. Intenta conectar con el gestor de colas
   - 2. Abre la cola SYSTEM.DEFAULT.LOCAL.QUEUE, coloca un mensaje en la cola, obtiene un mensaje de la cola y cierra la cola
   - 3. Desconecta del gestor de colas
   - 4. Devuelve un mensaje si las operaciones se realizan correctamente

A continuación, se muestra un ejemplo de los mensajes de petición y las respuestas que puede que vea. Los mensajes de solicitud reales y las respuestas dependen de la red IBM MQ.

```
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 ...
```

Nota:

1. z/OS En z/OS, deje el campo en blanco en la solicitud marcada (1).
2. Si elige la conexión de servidor, no verá los mensajes de solicitud marcados (2).
3. IBM i En IBM i, solo puede emitir el mandato `java MQIVP` desde QShell. Como alternativa, puede ejecutar la aplicación utilizando el mandato de lenguaje de control `RUNJVA CLASS(MQIVP)`.

Utilización de las aplicaciones de ejemplo IBM MQ classes for Java

Las aplicaciones de ejemplo IBM MQ classes for Java proporcionan una descripción general de las características comunes de la API IBM MQ classes for Java. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Acerca de esta tarea

Si necesita ayuda para crear sus propias aplicaciones, puede utilizar las aplicaciones de ejemplo como punto de partida. Se proporcionan ambas versiones, la de origen y la compilada, para cada aplicación. Revise el código fuente de ejemplo e identifique los pasos clave para crear cada objeto necesario para la aplicación (MQQueueManager, MQConstants , MQMessage, MQPutMessageOptions y MQDestination), y para establecer cualquier propiedad específica necesaria para especificar cómo desea que funcione la aplicación. Para obtener más información, consulte “Escritura de aplicaciones de IBM MQ classes for Java” en la página 382. Los ejemplos podrían estar sujetos a cambios en futuros releases de IBM MQ Java.

Tabla 55 en la página 379 muestra dónde se instalan las aplicaciones de ejemplo de IBM MQ classes for Java en cada plataforma:

Tabla 55. Directorios de instalación para las aplicaciones de ejemplo IBM MQ classes for Java

Plataforma	**Directorio**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\wmqjava\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/wmqjava/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/wmqjava`

Tabla 56 en la página 379 muestra los conjuntos de aplicaciones de ejemplo que se proporcionan con IBM MQ classes for Java.

Tabla 56. IBM MQ classes for Java aplicaciones de ejemplo

Nombre de ejemplo	**Descripción**
IMSBridgeSample.java	Programa de ejemplo para ilustrar el uso del puente IMS con IBM MQ classes for Java.
MQIVP.java	Programa de verificación de instalación de IBM MQ Java.
MQMessagePropertiesSample.java	Demuestra el uso de la API de propiedades de mensajes.
MQPubSubApiSample.java	Demuestra el uso de la API de publicación/suscripción.
MQSample.java	Programa de ejemplo para ilustrar cómo se coloca y se obtiene un mensaje de una cola.
MQSampleMessageManager.java	Clase de programa de utilidad para el manejo de mensajes en los ejemplos IBM MQ base Java .
mqjcivp.properties	Este paquete de recursos contiene los mensajes utilizados por el programa de verificación de instalación de IBM MQ classes for Java (`MQIVP.java`).

IBM MQ classes for Java proporcionan un script llamado `runjms` que se puede utilizar para ejecutar las aplicaciones de ejemplo. Este script configura el entorno IBM MQ para permitirle ejecutar las aplicaciones de ejemplo IBM MQ classes for Java.

Tabla 57 en la página 380 muestra la ubicación del script en cada plataforma:

Tabla 57. Ubicación del script `runjms`

Plataforma	Directorio
AIX Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	o
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

Para utilizar el script `runjms` para invocar una aplicación de ejemplo, complete los pasos siguientes:

Procedimiento

1. Abra un indicador de mandatos y vaya hasta el directorio que contiene la aplicación de ejemplo que desea ejecutar.
2. Escriba el mandato siguiente:

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   La aplicación de ejemplo muestra una lista de los parámetros que necesita.
3. Especifique el mandato siguiente para ejecutar el ejemplo con estos parámetros:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Ejemplo

Linux Por ejemplo, para ejecutar el ejemplo MQIVP en Linux, especifique los mandatos siguientes:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Conceptos relacionados

"¿Qué se instala para IBM MQ classes for JMS?" en la página 92
Se crean varios archivos y directorios cuando se instala IBM MQ classes for JMS. En Windows, se realizan algunas configuraciones durante la instalación estableciendo automáticamente variables de entorno. En otras plataformas, y en algunos entornos Windows, debe establecer variables de entorno para poder ejecutar aplicaciones de IBM MQ classes for JMS.

Resolución de problemas de IBM MQ classes for Java

Inicialmente, ejecute el programa de verificación de la instalación. Puede también ser necesario utilizar el recurso de rastreo.

Si una aplicación no finaliza correctamente, ejecute el programa de verificación de la instalación y siga los consejos que se proporcionan en los mensajes de diagnóstico. El programa de verificación de la instalación se describe en "Verificación de la instalación de IBM MQ classes for Java" en la página 377.

Si los problemas persisten y necesita consultar al servicio técnico de IBM, le pueden solicitar que active el recurso de rastreo. Para ello, haga lo siguiente.

Para rastrear el programa `MQIVP`:

- Cree un archivo de propiedades `com.ibm.mq.commonservices` (consulte Utilización de com.ibm.mq.commonservices.
- Escriba el mandato siguiente:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

donde:

- *commonservices_properties_file* es la vía de acceso (incluido el nombre de archivo) al archivo de propiedades `com.ibm.mq.commonservices` .
- *vía_acceso_biblioteca* es la vía de acceso de las bibliotecas de IBM MQ classes for Java (consulte “IBM MQ classes for Java bibliotecas” en la página 367).

Para obtener más información sobre cómo utilizar el rastreo, consulte Rastreo de aplicaciones IBM MQ classes for Java.

z/OS MQ Adv. VUE Conectividad de cliente Java con aplicaciones por lotes que se ejecutan en z/OS

En determinadas condiciones, una aplicación IBM MQ classes for Java en z/OS puede conectarse a un gestor de colas en z/OS utilizando una conexión de cliente. El uso de una conexión de cliente puede simplificar las topologías de IBM MQ .

Utilizando una conexión de cliente, si una aplicación IBM MQ classes for Java se está ejecutando en un entorno de proceso por lotes, y se aplica una de las condiciones siguientes, la aplicación puede conectarse a un gestor de colas z/OS remoto:

- LTS V 9.3.4 El código IBM MQ classes for Java es IBM MQ 9.3.4 o posterior, o Long Term Support con el APAR PH56722 aplicado. El gestor de colas puede estar en cualquier versión soportada.
- El gestor de colas al que se está conectando se está ejecutando con la titularidad de IBM MQ Advanced for z/OS Value Unit Edition y, por lo tanto, tiene el parámetro **ADVCAP** establecido en ENABLED. El gestor de colas puede estar en cualquier versión soportada.

Para obtener más información sobre IBM MQ Advanced for z/OS Value Unit Edition consulte Identificadores de producto deIBM MQ e información de exportación.

Consulte DISPLAY QMGR para obtener más información sobre **ADVCAP** y START QMGR para obtener más información sobre **QMGRPROD**.

Una aplicación IBM MQ classes for Java en z/OS no puede utilizar una conexión en modalidad de cliente para conectarse a un gestor de colas que no se está ejecutando en z/OS

Si una aplicación IBM MQ classes for Java en z/OS intenta conectarse utilizando la modalidad de cliente, y no se le permite hacerlo, se devuelve MQRC_ENVIRONMENT_ERROR .

Soporte de Advanced Message Security (AMS)

Las aplicaciones cliente IBM MQ classes for Java pueden utilizar AMS al conectarse a gestores de colas z/OS remotos, sujetos a las condiciones descritas anteriormente en este tema.

Para utilizar AMS de este modo, las aplicaciones cliente deben utilizar un tipo de almacén de claves de `jceracfks` en `keystore.conf`, donde:

- El prefijo de nombre de propiedad es `jceracfks`, en el que no se distingue entre mayúsculas y minúsculas.
- El almacén de claves es un conjunto de claves RACF.

- Las contraseñas no son necesarias y se ignoran. Esto se debe a que los conjuntos de claves RACF no utilizan contraseñas.
- Si se especifica el proveedor, este tiene que ser `IBMJCE`.

Cuando se utiliza `jceracfks` con AMS, el almacén de claves debe tener el formato: `safkeyring://` *`user`*`/`*`keyring`*, donde:

- `safkeyring` es un literal en el que no se distingue entre mayúsculas y minúsculas.
- *`user`* es el ID de usuario de RACF que es propietario del conjunto de claves
- *`keyring`* es el nombre del conjunto de claves RACF y el nombre del conjunto de claves distingue entre mayúsculas y minúsculas

En el ejemplo siguiente se utiliza el conjunto de claves AMS estándar para el usuario `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Conceptos relacionados

"Conectividad de cliente JMS/Jakarta Messaging con aplicaciones por lotes que se ejecutan en z/OS" en la página 130

En determinadas condiciones, una aplicación IBM MQ classes for JMS/Jakarta Messaging en z/OS puede conectarse a un gestor de colas en z/OS utilizando una conexión de cliente. El uso de una conexión de cliente puede simplificar las topologías de IBM MQ .

Escritura de aplicaciones de IBM MQ classes for Java

Esta colección de temas proporciona información para ayudarle a escribir aplicaciones Java para interactuar con sistemas IBM MQ.

Para utilizar IBM MQ classes for Java para acceder a colas de IBM MQ, debe escribir programas Java que contengan llamadas para transferir mensajes a colas de IBM MQ y obtener mensajes de ellas. Para obtener detalles de clases individuales, consulte IBM MQ classes for Java.

Nota: IBM MQ classes for Java no da soporte a la reconexión automática del cliente.

La interfaz de IBM MQ classes for Java

La interfaz de programación de aplicaciones de IBM MQ orientada a procedimientos utiliza verbos que actúan sobre objetos. La interfaz de programación de Java utiliza objetos, sobre los cuales el usuario actúa mediante la llamada a métodos.

La interfaz de programación de aplicaciones de IBM MQ orientada a procedimientos está basada en verbos, tales como los siguientes:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Estos verbos toman todos como parámetro un descriptor de contexto del objeto de IBM MQ sobre el que tienen que trabajar. El programa del usuario consta de un conjunto de objetos de IBM MQ, sobre los que el usuario actúa invocando métodos sobre esos objetos.

Cuando utiliza la interfaz orientada a procedimientos, se desconecta del gestor de colas utilizando la llamada MQDISC(Hconn, CompCode, Reason), donde *Hconn* es un descriptor de contexto del gestor de colas.

En la interfaz Java, el gestor de colas está representado por un objeto de la clase MQQueueManager. Puede desconectarse del gestor de colas invocando el método disconnect() en esa clase.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
```

```
...
// disconnect from the queue manager
queueManager.disconnect();
```

Modalidades de conexión de IBM MQ classes for Java

La forma en que programa para IBM MQ classes for Java tiene algunas dependencias respecto de las modalidades de conexión que desee utilizar.

Si utiliza conexiones de cliente, hay varias diferencias respecto del IBM MQ MQI client, pero es conceptualmente similar. Si utiliza la modalidad de enlaces, puede utilizar enlaces de vía de acceso rápida y puede emitir el mandato MQBEGIN. Puede especificar la modalidad que se debe utilizar estableciendo variables en la clase MQEnvironment.

Conexiones de cliente de IBM MQ classes for Java

Cuando IBM MQ classes for Java se utiliza como cliente, es similar al IBM MQ MQI client, pero existen varias diferencias.

Si está programando para *IBM MQ classes for Java* para utilizarlo como cliente, tenga en cuenta las diferencias siguientes:

- Sólo ofrece soporte para TCP/IP.
- No lee ninguna variable de entorno de IBM MQ durante el proceso de inicio.
- La información que se almacenaría en una definición de canal y en variables de entorno se puede almacenar en una clase denominada Environment. Como alternativa, esta información se puede pasar como parámetros cuando se establece la conexión.
- Las condiciones de error y de excepción se escriben en un archivo de registro especificado en la clase `MQException`. El destino predeterminado para los errores es la consola de Java.
- Sólo los atributos siguientes contenidos en un archivo de configuración de cliente de IBM MQ son aplicables a IBM MQ classes for Java. Si especifica otros atributos, no serán efectivos.

Stanza	**Atributo**
Stanza ClientExitPath del archivo de configuración de cliente	Vía de acceso predeterminada de las salidas
Stanza ClientExitPath del archivo de configuración de cliente	ExitsDefaultPath64
Stanza ClientExitPath del archivo de configuración de cliente	JavaExitsClasspath
Stanza MessageBuffer del archivo de configuración de cliente	MaximumSize
Stanza MessageBuffer del archivo de configuración de cliente	PurgeTime
Stanza MessageBuffer del archivo de configuración de cliente	UpdatePercentage
Stanza TCP del archivo de configuración de cliente	ClntRcvBuffSize
Stanza TCP del archivo de configuración de cliente	ClntSndBuffSize
Stanza TCP del archivo de configuración de cliente	Connect_Timeout
Stanza TCP del archivo de configuración de cliente	KeepAlive

- Si se conecta a un gestor de colas que necesita que se conviertan datos de tipo carácter, el cliente de Java V7 es ahora capaz de realizar la conversión si el gestor de colas no es capaz de hacerlo. La JVM de cliente debe permitir la conversión entre el CCSID del cliente y el CCSID del gestor de colas.
- La reconexión automática de cliente no está soportada en IBM MQ classes for Java.

Cuando se utiliza en la modalidad de cliente, *IBM MQ classes for Java* no admite la llamada MQBEGIN.

Modalidad de enlaces de IBM MQ classes for Java
La modalidad de enlaces de IBM MQ classes for Java difiere de la modalidad de cliente de tres maneras principales.

Cuando se utiliza en la modalidad de enlaces, IBM MQ classes for Java utiliza la interfaz nativa de Java (JNI) para llamar directamente en la API del gestor de colas existente, en lugar de comunicarse a través de una red.

De forma predeterminada, las aplicaciones que utilizan las IBM MQ classes for Java en la modalidad de enlaces se conectan a un gestor de colas utilizando el valor MQCNO_STANDARD_BINDINGS para *ConnectOption*.

Las IBM MQ classes for Java soportan los valores siguientes para *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

Para obtener más información sobre *ConnectOptions*, consulte "Conexión a un gestor de colas mediante la llamada MQCONNX" en la página 756.

La modalidad de enlaces admite la llamada MQBEGIN para iniciar unidades de trabajo globales que se coordinan mediante el gestor de colas, en todas las plataformas, salvo IBM MQ for IBM i y IBM MQ for z/OS.

La mayoría de los parámetros proporcionados por la clase `MQEnvironment` no son aplicables a la modalidad de enlaces y no se tienen en cuenta.

Definición de la conexión de IBM MQ classes for Java que se debe utilizar
El tipo de conexión que se debe utilizar está determinado por el valor de las variables contenidas en la clase MQEnvironment.

Se utilizan dos variables:

MQEnvironment.properties
El tipo de conexión está determinado por el valor asociado al nombre de clave `CMQC.TRANSPORT_PROPERTY`. Los valores posibles son los siguientes:

CMQC.TRANSPORT_MQSERIES_BINDINGS
Conexión en modalidad de enlaces

CMQC.TRANSPORT_MQSERIES_CLIENT
Conexión en modalidad de cliente

CMQC.TRANSPORT_MQSERIES
La modalidad de conexión viene determinada por el valor de la propiedad *hostname*

MQEnvironment.hostname
Establezca el valor de esta variable de la forma siguiente:

- Para las conexiones de cliente, establezca el valor de esta variable en el nombre de host del servidor de IBM MQ con el que desee conectar
- Para la modalidad de enlaces, no establezca esta variable, o establezca su valor en nulo

Operaciones en gestores de colas

Esta colección de temas describe cómo conectar con un gestor de colas y desconectarse de él utilizando IBM MQ classes for Java.

Configuración del entorno de IBM MQ para IBM MQ classes for Java
Para que una aplicación se pueda conectar a un gestor de colas en la modalidad de cliente, la aplicación debe especificar el nombre de canal, el nombre de host y el número de puerto.

Nota: La información de este tema sólo es pertinente si la aplicación se conecta a un gestor de colas en la modalidad de cliente. La información no es aplicable si la aplicación se conecta en la modalidad de enlaces. Consulte: “Modalidades de conexión para IBM MQ classes for JMS” en la página 114

Puede especificar el nombre de canal, el nombre de host y el número de puerto de una de las dos maneras siguientes: como campos de la clase MQEnvironment o como propiedades del objeto MQQueueManager.

Si define campos en la clase MQEnvironment, se aplican a toda la aplicación, excepto cuando prevalecen los valores de una tabla hash de propiedades. Para especificar el nombre de canal y el nombre del host en MQEnvironment, utilice el código siguiente:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

Esto es equivalente a definir una variable de entorno **MQSERVER**:

```
"java.client.channel/TCP/host.domain.com".
```

De forma predeterminada, los clientes de Java intentan conectar con un escucha de IBM MQ en el puerto 1414. Para especificar un puerto diferente, utilice el código siguiente:

```
MQEnvironment.port = nnnn;
```

donde nnnn es el número de puerto necesario

Si pasa propiedades a un objeto de gestor de colas cuando lo crea, sólo se aplicarán a dicho gestor de colas. Cree entradas en un objeto Hashtable con claves de **hostname**, **channel**, y, opcionalmente, **port**, junto con valores adecuados. Para utilizar el puerto predeterminado, 1414, puede omitir la entrada **port**. Cree el objeto MQQueueManager utilizando un constructor que acepte como entrada la tabla hash de propiedades.

Identificación de una conexión para el gestor de colas estableciendo un nombre de aplicación

Una aplicación puede definir un nombre que identifique su conexión al gestor de colas. Este nombre de aplicación se muestra mediante el mandato **DISPLAY CONN MQSC/PCF** (donde el campo se denomina **APPLTAG**) o en la pantalla **Conexiones de aplicación** de IBM MQ Explorer (donde el campo se denomina **App name**).

Los nombres de aplicación están limitados a 28 caracteres, por lo que los nombres más largos se truncan. Si no se especifica un nombre de aplicación, se proporciona un valor predeterminado. El nombre predeterminado está basado en la clase invocadora (main), pero si esta información no está disponible, se utiliza el texto `Cliente IBM MQ para Java`.

Si se utiliza el nombre de la clase invocadora, se ajusta a la longitud disponible eliminando los nombres de paquete iniciales, si es necesario. Por ejemplo, si la clase invocadora es `com.example.MainApp`, se utiliza el nombre completo, pero si la clase invocadora es `com.example.dictionaryAndThesaurus.multilingual.mainApp`, se utiliza el nombre `multilingual.mainApp`, porque es la combinación más larga formada por el nombre de clase y el nombre de paquete situado más a la derecha que se ajusta a la longitud disponible.

Si el nombre de clase propiamente dicho tiene más de 28 caracteres de longitud, se trunca para que quepa. Por ejemplo, `com.example.mainApplicationForSecondTestCase` pasa a ser `mainApplicationForSecondTest`.

Para definir un nombre de aplicación en la clase MQEnvironment, añada el nombre a la tabla hash MQEnvironment.properties, con una clave **MQConstants.APPNAME_PROPERTY**, utilizando el código siguiente:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Para definir un nombre de aplicación en la tabla hash de propiedades que se pasa al constructor MQQueueManager, añada el nombre a la tabla hash de propiedades con una clave **MQConstants.APPNAME_PROPERTY**.

Alteración temporal de las propiedades especificadas en un archivo de configuración de cliente de IBM MQ

Un archivo de configuración de cliente de IBM MQ también puede especificar las propiedades que se utilizan para configurar IBM MQ classes for Java. Pero las propiedades especificadas en un archivo de configuración de IBM MQ MQI client se aplican sólo cuando una aplicación se conecta a un gestor de colas en la modalidad de cliente.

Si es necesario, puede alterar temporalmente cualquier atributo en un archivo de configuración de IBM MQ de cualquiera de las formas siguientes. Las opciones se muestran en orden de prioridad.

- Establezca una propiedad del sistema Java para la propiedad de configuración.
- Defina la propiedad en la correlación MQEnvironment.properties.
- En Java 5 y releases posteriores, establezca una variable de entorno del sistema.

Sólo los atributos siguientes contenidos en un archivo de configuración de cliente de IBM MQ son aplicables a IBM MQ classes for Java. Si especifica o altera temporalmente otros atributos, dicha acción no tendrá afecto.

Stanza	Atributo
Stanza ClientExitPath del archivo de configuración de cliente	Vía de acceso predeterminada de las salidas
Stanza ClientExitPath del archivo de configuración de cliente	ExitsDefaultPath64
Stanza ClientExitPath del archivo de configuración de cliente	JavaExitsClasspath
Stanza MessageBuffer del archivo de configuración de cliente	MaximumSize
Stanza MessageBuffer del archivo de configuración de cliente	PurgeTime
Stanza MessageBuffer del archivo de configuración de cliente	UpdatePercentage
Stanza TCP del archivo de configuración de cliente	ClntRcvBufSize
Stanza TCP del archivo de configuración de cliente	ClntSndBufSize
Stanza TCP del archivo de configuración de cliente	Connect_Timeout
Stanza TCP del archivo de configuración de cliente	KeepAlive

Conexión a un gestor de colas en IBM MQ classes for Java
Conéctese a un gestor de colas creando una instancia nueva de la clase MQQueueManager. Desconéctese de un gestor de colas llamando al método disconnect().

Ahora está preparado para conectarse a un gestor de colas creando una nueva instancia de la clase MQQueueManager:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Para desconectarse de un gestor de colas, invoque el método disconnect() en el gestor de colas:

```
queueManager.disconnect();
```

Cuando invoca el método de desconexión, se cierran todas las colas y los procesos abiertos a los que ha accedido mediante ese gestor de colas. Pero es una práctica de programación recomendada cerrar esos recursos explícitamente cuando termine de utilizarlos. Para ello, utilice el método close() sobre los objetos pertinentes.

Los métodos commit() y backout() ejecutados sobre un gestor de colas equivalen a las llamadas MQCMIT y MQBACK que se utilizan en la interfaz de procedimientos.

Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for Java
Una aplicación cliente de IBM MQ classes for Java puede utilizar las definiciones de canal de conexión de cliente almacenadas en una tabla de definición de canal de cliente (CCDT).

Como alternativa a la creación de una definición de canal de conexión de cliente estableciendo determinados campos y propiedades de entorno en la clase `MQEnvironment` o pasándolos a un `MQQueueManager` en una tabla hash de propiedades, una aplicación cliente de IBM MQ classes for Java puede utilizar las definiciones de canal de conexión de cliente que están almacenadas en una tabla de definición de canal de cliente. Estas definiciones se crean mediante los mandatos IBM MQ Script (MQSC) o IBM MQ Programmable Command Format (PCF), o utilizando IBM MQ Explorer.

Cuando la aplicación crea un objeto `MQQueueManager`, el cliente de IBM MQ classes for Java busca en la tabla de definición de canal de cliente una definición de canal adecuada para iniciar un canal MQI. Para obtener más información sobre las tablas de definición de canal de cliente y sobre cómo construir una, consulte Tabla de definición de canal de cliente.

Para utilizar una tabla de definición de canal de cliente, debe crear primero un objeto de URL. El objeto de URL encapsula un URL (localizador uniforme de recursos) que identifica el nombre y la ubicación del archivo que contiene la tabla de definición de canal de cliente y especifica cómo se puede acceder al archivo.

Por ejemplo, si el archivo `ccdt1.tab` contiene una tabla de definición de canal de cliente y está almacenado en el mismo sistema en el que se ejecuta la aplicación, la aplicación puede crear un objeto de URL de la forma siguiente:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Por ejemplo, suponga que el archivo `ccdt2.tab` contiene una tabla de definición de canal de cliente y está almacenado en un sistema distinto de aquel en el se ejecuta la aplicación. Si se puede acceder al archivo utilizando el protocolo FTP, la aplicación puede crear un objeto de URL de la manera siguiente:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Una vez que la aplicación ha creado un objeto de URL, la aplicación puede crear un objeto `MQQueueManager` utilizando uno de los constructores que utiliza un objeto de URL como parámetro. He aquí un ejemplo:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Esta sentencia hace que el cliente de IBM MQ classes for Java acceda a la tabla de definición de canal de cliente identificada por el objeto de URL chanTab2, busque en la tabla una definición de canal de conexión de cliente adecuada y utilice la definición de canal para iniciar un canal MQI en el gestor de colas llamado MARS.

Rigen las consideraciones siguientes si una aplicación utiliza una tabla de definición de canal de cliente:

- Cuando la aplicación crea un objeto `MQQueueManager` utilizando un constructor que toma un objeto de URL como parámetro, no se debe establecer ningún nombre de canal en la clase `MQEnvironment`, ya sea como un campo o como una propiedad de entorno. Si se establece un nombre de canal, el cliente de IBM MQ classes for Java emite una `MQException`. Se considera que hay establecido un nombre de canal en el campo o propiedad de entorno si su valor es distinto de nulo, una serie vacía o una serie en blanco.
- El parámetro **queueManagerName** en el constructor `MQQueueManager` puede tener uno de los valores siguientes:
  - El nombre de un gestor de colas
  - Un asterisco (*) seguido por el nombre de un grupo de gestores de colas.
  - Un asterisco (*)
  - Nulo, una serie vacía o una serie que tenga todos los caracteres en blanco

  Estos son los mismos valores que se pueden utilizar para el parámetro **QMgrName** en una llamada MQCONN emitida por una aplicación cliente que está utilizando Interfaz de Colas de Mensajes (MQI). Para obtener más información sobre el significado de estos valores, consulte"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740.

  Si la aplicación utiliza la técnica de agrupación de conexiones, consulte "Control de la agrupación de conexiones predeterminada en IBM MQ classes for Java" en la página 408.
- Cuando el cliente de IBM MQ classes for Java encuentra una definición de canal de conexión de cliente adecuada en la tabla de definición de canal de cliente, sólo utiliza la información extraída de esta definición de canal para iniciar un canal MQI. No se tienen en cuenta los campos o propiedades de entorno relacionados con el canal que la aplicación pueda haber establecido en la clase `MQEnvironment`.

  En particular, tenga en cuenta los puntos siguientes si está utilizando Transport Layer Security (TLS):
  - Un canal MQI sólo utiliza TLS si la definición de canal extraída de la tabla de definición de canal de cliente especifica el nombre de una CipherSpec que está soportada por el cliente de IBM MQ classes for Java.
  - Una tabla de definición de canal de cliente también contiene información sobre la ubicación de los servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados (listas CRL). El cliente de IBM MQ classes for Java sólo utiliza esta información para acceder a servidores LDAP que contienen las listas CRL.
  - Una tabla de definición de canal de cliente también puede contener la ubicación de un programa de respuesta OCSP. IBM MQ classes for Java no puede utilizar la información OCSP en un archivo de tabla de definición de canal de cliente. Sin embargo, puede configurar OCSP, tal como se describe en la sección Utilización de Online Certificate Protocol

  Para obtener más información sobre la utilización de TLS con una tabla de definición de canal de cliente, consulte Especificación de que un canal MQI utiliza TLS.

  Tenga también en cuenta los puntos siguientes si está utilizando salidas de canal:
  - Un canal MQI utiliza las salidas de canal y los datos de usuario asociados que están especificados por la definición de canal extraída de la tabla de definición de canal de cliente en preferencia a las salidas de canal y los datos especificados utilizando otros métodos.
  - Una definición de canal extraída de una tabla de definiciones de canal de cliente puede especificar salidas de canal escritas en Java, C o C++. Para obtener más información sobre cómo escribir una salida de canal en Java , consulte"Creación de una salida de canal en IBM MQ classes for Java" en la página 401. Para obtener más información sobre cómo escribir una salida de canal en otros idiomas, consulte el apartado "Utilización de salidas de canal no escritas en Java con IBM MQ classes for Java" en la página 405.

Especificación de un rango de puertos para las conexiones de cliente de IBM MQ classes for Java
Puede especificar un puerto, o rango de puertos, al que una aplicación se puede enlazar, de una de dos maneras.

Cuando una aplicación de IBM MQ classes for Java intenta conectar con un gestor de colas de IBM MQ en la modalidad de cliente, un cortafuegos podría permitir solo las conexiones que se originan en un puerto o rango de puertos especificado. En esta situación, puede especificar un puerto, o bien un rango de puertos, a la que la aplicación se puede enlazar. Puede especificar los puertos de las formas siguientes:

- Puede establecer el campo localAddressSetting en la clase MQEnvironment. He aquí un ejemplo:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- Puede establecer la propiedad de entorno CMQC.LOCAL_ADDRESS_PROPERTY. He aquí un ejemplo:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Cuando puede construir el objeto MQQueueManager, puede pasar una tabla hash de propiedades que contenga LOCAL_ADDRESS_PROPERTY con el valor "192.0.2.0(2000,3000)".

En cada uno de estos ejemplos, cuando la aplicación se conecta posteriormente a un gestor de colas, la aplicación se enlaza con una dirección IP local y un número de puerto situado en el rango de 192.0.2.0(2000) a 192.0.2.0(3000).

En un sistema con más de una interfaz de red, también puede utilizar el campo localAddressSetting, o la propiedad de entorno CMQC.LOCAL_ADDRESS_PROPERTY para especificar qué interfaz de red se debe utilizar para una conexión.

Pueden producirse errores de conexión si restringe el rango de puertos. Si se produce un error, se emite una excepción MQException que contiene el código de razón MQRC_Q_MGR_NOT_AVAILABLE de IBM MQ y el mensaje siguiente:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Puede producirse un error si se están utilizando todos los puertos del rango especificado o si la dirección IP, el nombre de host o el número de puerto especificados no son válidos (un número de puerto negativo, por ejemplo).

Acceso a colas, temas y procesos en IBM MQ classes for Java

Para acceder a colas, temas y procesos, utilice métodos de la clase MQQueueManager. MQOD (estructura de descriptor de objeto) se ha contraído en los parámetros de estos métodos.

Colas

Para abrir una cola, puede utilizar el método accessQueue de la clase MQQueueManager. Por ejemplo, en un gestor de colas denominado queueManager, utilice el código siguiente:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

El método accessQueue devuelve un nuevo objeto de la clase MQQueue.

Cuando haya terminado de utilizar la cola, utilice el método close() para cerrarla, tal como se muestra en el ejemplo siguiente:

```
queue.close();
```

También puede crear una cola utilizando el constructor MQQueue. Los parámetros son exactamente los mismos que para el método accessQueue, con la adición de un parámetro de gestor de colas. Por ejemplo:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
```

```
                                CMQC.MQOO_OUTPUT,
                                "qMgrName",
                                "dynamicQName",
                                "altUserID");
```

Puede especificar varias opciones al crear colas. Para obtener detalles sobre estas opciones, consulte Class.com.ibm.mq.MQQueue. La creación de un objeto de cola utilizando este método permite escribir sus propias subclases de MQQueue.

Temas

Similarmente, puede abrir un tema utilizando el método accessTopic de la clase MQQueueManager. Por ejemplo, en un gestor de colas denominado queueManager, utilice el código siguiente para crear un suscriptor y un publicador:

```
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);
```

Cuando haya terminado de utilizar el tema, utilice el método close() para cerrarlo.

También puede crear un tema utilizando el constructor MQTopic. Los parámetros son exactamente los mismos que para el método accessTopic, con la adición de un parámetro de gestor de colas. Por ejemplo:

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

Puede especificar varias opciones al crear temas. Para obtener detalles sobre estas opciones, consulte Class com.ibm.mq.MQTopic. El crear un objeto de tema de esta manera le permite escribir sus propias subclases de MQTopic.

Un tema se debe abrir para publicación o suscripción. La clase MQQueueManager tiene ocho métodos accessTopic y la clase Topic tiene ocho constructores. En cada caso, cuatro de estos tienen un parámetro **destination** y cuatro tienen un parámetro **subscriptionName** (incluido dos que tienen ambos). Sólo se pueden utilizar para abrir el tema para suscripciones. Los dos métodos restantes tiene un parámetro **openAs** y se puede abrir el tema para publicación o suscripción dependiendo del valor del parámetro **openAs**.

Para crear un tema como un suscriptor duradero, utilice un método accessTopic de la clase MQQueueManager o un constructor MQTopic que acepte un nombre de suscripción, en cualquiera de los casos, establezca la opción CMQC.MQSO_DURABLE.

todos los Procesos

Para acceder a un proceso, utilice el método accessProcess de MQQueueManager. Por ejemplo, en un gestor de colas llamado queueManager, utilice el código siguiente para crear un objeto MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Para acceder a un proceso, utilice el método accessProcess de MQQueueManager.

El método accessProcess devuelve un nuevo objeto de la clase MQProcess.

Cuando haya terminado de utilizar el objeto de proceso, utilice el método close() para cerrarlo, tal como se muestra en el ejemplo siguiente:

```
process.close();
```

También puede crear un proceso utilizando el constructor MQProcess. Los parámetros son exactamente los mismos que para el método accessProcess, con la adición de un parámetro de gestor de colas. Por ejemplo:

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

La creación de un objeto de proceso mediante este método le permite crear sus propias subclases de MQProcess.

Manejo de mensajes en IBM MQ classes for Java

Los mensajes se representan mediante la clase MQMessage. Puede transferir y obtener mensajes mediante métodos de la clase MQDestination, que tiene subclases de MQQueue y MQTopic.

Puede transferir mensajes a colas o temas utilizando el método put() de la clase MQDestination. Puede obtener mensajes de colas o temas utilizando el método get() de la clase MQDestination. A diferencia de la interfaz de procedimientos, donde MQPUT y MQGET transfieren y obtienen matrices de bytes, el lenguaje de programación Java transfiere y obtiene instancias de la clase MQMessage. La clase MQMessage encapsula el almacenamiento intermedio de datos que contiene los datos del mensaje, junto con todos los parámetros de MQMD (descriptor de mensaje) y las propiedades de mensaje que describen ese mensaje.

Para crear un nuevo mensaje, cree una nueva instancia de la clase MQMessage y utilice los métodos writeXXX para transferir datos al almacenamiento intermedio de mensajes.

Cuando se crea la instancia de mensaje nueva, todos los parámetros MQMD se establecen automáticamente en sus valores predeterminados, tal como se define en Valores iniciales y declaraciones de lenguaje para MQMD. El método put() de MQDestination también toma una instancia de la clase MQPutMessageOptions como parámetro. Esta clase representa la estructura MQPMO. En el ejemplo siguiente se crea un mensaje y se transfiere a una cola:

```
// 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);
```

El método get() de MQDestination devuelve una instancia nueva de MQMessage, que representa el mensaje recién obtenido de la cola. También toma una instancia de la clase MQGetMessageOptions como parámetro. Esta clase representa la estructura MQGMO.

No es necesario especificar un tamaño máximo de mensaje, pues el método get() ajusta automáticamente el tamaño de su almacenamiento intermedio interno para dar cabida al mensaje entrante. Utilice los métodos readXXX de la clase MQMessage para acceder a los datos del mensaje devuelto.

El ejemplo siguiente muestra cómo obtener un mensaje de una cola:

```
// 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);
```

Puede modificar el formato numérico que utilizan los métodos de lectura y escritura estableciendo la variable de miembro *encoding*.

Puede modificar el juego de caracteres que se va a utilizar para leer y escribir series de caracteres estableciendo la variable de miembro *characterSet*.

Consulte Clase MQMessage para obtener más información.

Nota: El método writeUTF() de MQMessage codifica automáticamente la longitud de la serie así como los bytes Unicode que contiene. Cuando el mensaje será leído por otro programa de Java (mediante ReadUTF()), esta es la forma más sencilla de enviar información de tipo serie.

Mejora del rendimiento de los mensajes no persistentes en IBM MQ classes for Java
Para mejorar el rendimiento cuando se examinan mensajes o se consumen mensajes no persistentes de una aplicación de cliente, puede utilizar la *lectura anticipada*. Las aplicaciones cliente que utilicen el consumo asíncrono o MQGET se beneficiarán de las mejoras en el rendimiento cuando examinen mensajes o consuman mensajes no persistentes.

Para obtener información general sobre el recurso de lectura anticipada, consulte el tema relacionado.

En IBM MQ classes for Java, utilice las propiedades CMQC.MQSO_READ_AHEAD y CMQC.MQSO_NO_READ_AHEAD de un objeto MQQueue o MQTopic para determinar si los consumidores de mensajes y los examinadores de colas pueden utilizar la lectura anticipada en dicho objeto.

Transferencia asíncrona de mensajes utilizando IBM MQ classes for Java
Para transferir un mensaje de forma asíncrona, establezca MQPMO_ASYNC_RESPONSE.

Los mensajes se transfieren a las colas o temas mediante el método put() de la clase MQDestination. Para transferir un mensaje de forma asíncrona, es decir, para permitir que la operación se ejecute sin tener que esperar una respuesta del gestor de colas, puede establecer MQPMO_ASYNC_RESPONSE en el campo de opciones de MQPutMessageOptions. Para determinar si las transferencias asíncronas se han realizado o no correctamente, utilice la llamada MQQueueManager.getAsyncStatus.

Publicación/suscripción en IBM MQ classes for Java
En IBM MQ classes for Java, el tema se representa mediante la clase MQTopic, y se publica utilizando los métodos MQTopic.put().

Para obtener información general sobre la publicación/suscripción de IBM MQ, consulte Mensajería de publicación/suscripción.

Manejo de cabeceras de mensaje de IBM MQ con IBM MQ classes for Java
Se proporcionan clases de Java que representan distintos tipos de cabecera de mensaje. También se proporcionan dos clases auxiliares.

La interfaz MQHeader

La interfaz MQHeader describe objetos de cabecera. Esta interfaz proporciona métodos de uso general para acceder a campos de cabecera y para leer y escribir el contenido de mensajes. Cada tipo de cabecera tiene su propia clase que implementa la interfaz MQHeader y añade los métodos getter y setter para campos individuales. Por ejemplo, el tipo de cabecera MQRFH2 se representa mediante la clase MQRFH2; el tipo de cabecera MQDLH por la clase MQDLH, etc. Las clases de cabecera realizan automáticamente cualquier conversión de datos y pueden leer o escribir datos con cualquier codificación numérica o juego de caracteres (CCSID) especificado.

Importante: Las clases de cabeceras MQRFH2 tratan el mensaje como un archivo de acceso aleatorio, por lo que el cursor debe estar situado al principio del mensaje. Antes de utilizar una clase de cabecera de mensaje interna como MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH o MQXMIT, asegúrese de actualizar la posición del cursor del mensaje a la ubicación correcta antes de pasar el mensaje a la clase.

Clases auxiliares

Las clases auxiliares MQHeaderIterator y MQHeaderList ayudan a leer y descodificar (analizar) el contenido de las cabeceras en los mensajes:

- La clase MQHeaderIterator trabaja como un java.util.Iterator. Mientras haya más cabeceras en el mensaje, el método next() devuelve true (verdadero) y el método nextHeader() o next() devuelve el siguiente objeto de cabecera.
- MQHeaderList funciona como una java.util.List. Al igual que MQHeaderIterator, analiza el contenido de la cabecera, pero también le permite buscar cabeceras determinadas, añadir cabeceras nuevas, eliminar cabeceras existentes, actualizar campos de cabecera y luego escribir el contenido de la cabecera en un mensaje. Como alternativa, puede crear una MQHeaderList vacía, después llenarla con instancias de cabecera y escribirla en un mensaje una sola vez o repetidamente.

Las clases MQHeaderIterator y MQHeaderList utilizan la información de MQHeaderRegistry para saber qué clases de cabecera de IBM MQ están asociadas con determinados tipos de mensajes y formatos. MQHeaderRegistry se configura con el conocimiento de todos los formatos y tipos de cabecera actuales de IBM MQ y sus clases de implementación, y el usuario también puede registrar sus propios tipos de cabecera.

Se proporciona soporte para las siguientes cabeceras de IBM MQ de uso habitual

- MQRFH - Cabecera de reglas y formato
- MQRFH2 - Al igual que MQRFH, se utiliza para intercambiar mensajes con un intermediario de mensajes perteneciente a IBM Integration Bus. También se utiliza para contener propiedades de mensaje
- MQCIH - Puente de CICS
- MQDLH - Cabecera de mensajes no entregados
- MQIIH – Cabecera información de IMS
- MQRMH - Cabecera de mensaje de referencia
- MQSAPH - Cabecera SAP
- MQWIH - Cabecera de información de trabajo
- MQXQH - Cabecera de cola de transmisión
- MQDH - Cabecera de distribución
- MQEPH - Cabecera PCF encapsulada

También puede definir clases que representen sus propias cabeceras.

Para utilizar una clase MQHeaderIterator para obtener una cabecera RFH2, establezca MQGMO_PROPERTIES_FORCE_MQRFH2 en GetMessageOptions, o establezca la propiedad de cola PROPCTL en FORCE.

Visualización de todas las cabeceras de un mensaje en IBM MQ classes for Java
En este ejemplo, una instancia de MQHeaderIterator analiza las cabeceras de un MQMessage que se ha recibido de una cola. Los objetos de MQHeader devueltos desde el método nextHeader() visualizan su estructura y contenido cuando se invoca su método toString.

```
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);
}
```

Omisión de las cabeceras de un mensaje en IBM MQ classes for Java
En este ejemplo, el método skipHeaders() de MQHeaderIterator coloca el cursor de lectura del mensaje inmediatamente después de la última cabecera.

```
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 ();
```

Búsqueda del código de razón en un mensaje no entregado en utiliza IBM MQ classes for Java
En este ejemplo, el método read llena el objeto de MQDLH leyendo en el mensaje. Después de la operación de lectura, el cursor de lectura del mensaje se coloca inmediatamente después del contenido de la cabecera MQDLH.

Los mensajes de la cola de mensajes no entregados del gestor de colas tiene como prefijo una cabecera de mensaje no entregado (MQDLH). Para determinar cómo manejar esos mensajes, por ejemplo, para determinar si se deben recuperar o descartar, una aplicación de manejo de mensajes no entregados debe examinar el código de razón contenido en la MQDLH.

```
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 ());
```

Todas las clases de cabecera proporcionan también un constructor simplificado para inicializarse a sí mismas directamente desde el mensaje en un solo paso. Por lo tanto, el código de este ejemplo podría simplificarse como sigue:

```
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 ());
```

Lectura y eliminación de la cabecera de un mensaje no entregado en IBM MQ classes for Java
En este ejemplo, se utiliza MQDLH para eliminar la cabecera de un mensaje no entregado.

Normalmente, una aplicación de gestión de mensajes no entregados volverá a enviar los mensajes que se hayan rechazado si su código de razón indica un error transitorio. Antes de volver a someter el mensaje, debe eliminar la cabecera MQDLH.

Este ejemplo realiza los siguientes pasos (vea los comentarios en el código del ejemplo):

1. La MQHeaderList lee el mensaje completo y cada cabecera encontrada en el mensaje se convierte en un elemento de la lista.
2. Los mensajes no entregados contienen una MQDLH como primera cabecera, por lo que ésta puede encontrarse en el primer elemento de la lista de la cabecera. La MQDLH ya se ha rellenado a partir del mensaje al crear la MQHeaderList, por lo que no es necesario invocar su método de lectura.

3. El código de razón se extrae usando el método getReason() proporcionado por la clase MQDLH.
4. Se ha examinado el código de razón que indica que es adecuado volver a enviar el mensaje. La MQDLH se elimina usando el método MQHeaderList remove().
5. La MQHeaderList graba el resto del contenido en un nuevo objeto de mensaje. El nuevo mensaje contiene ahora todo lo del mensaje original excepto la MQDLH y puede grabarse en una cola. El argumento **true** para el constructor y para el método de grabación indica que el cuerpo del mensaje se ha de mantener dentro de la MQHeaderList y se ha de volver a grabar de nuevo.
6. El campo de formato del descriptor de mensaje del nuevo mensaje contiene ahora el valor que estaba anteriormente en el campo de formato de la MQDLH. Los datos del mensaje coinciden con la codificación numérica y el identificador de conjunto de caracteres codificados (CCSID) establecidos en el descriptor de mensaje.

```
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.
```

Visualización del contenido de un mensaje en IBM MQ classes for Java
Este ejemplo utiliza MQHeaderList para visualizar el contenido de un mensaje, incluidas sus cabeceras.

Los datos de salida muestran una vista del contenido de todas las cabeceras así como el cuerpo del mensaje. La clase MQHeaderList descodifica todas las cabeceras a la vez, mientras que MQHeaderIterator efectúa iteraciones sobre ellas bajo el control de la aplicación. Puede utilizar esta técnica para proporcionar una herramienta de depuración simple al escribir aplicaciones de WebSphere MQ.

```
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));
```

Este ejemplo también visualiza los campos de descriptor de mensaje, utilizando la clase MQMD. El método copyFrom() de la clase com.ibm.mq.headers.MQMD rellena el objeto de cabecera a partir de los campos del descriptor de mensaje de MQMessage en vez de hacerlo mediante la lectura del cuerpo del mensaje.

```
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));
```

Búsqueda de un tipo específico de cabecera en un mensaje en IBM MQ classes for Java
Este ejemplo utiliza el método indexOf(String) de MQHeaderList para buscar una cabecera MQRFH2 en un mensaje, si existe alguno.

```
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);
        ...
}
```

Análisis de una cabecera MQRFH2 en IBM MQ classes for Java
Este ejemplo muestra cómo acceder a un valor de campo conocido en una carpeta especificada, utilizando la clase MQRFH2.

La clase MQRFH2 proporciona varias formas de acceder no sólo a los campos de la parte fija de la estructura, sino también al contenido de la carpeta codificada en XML que se transporta dentro del campo NameValueData. Este ejemplo muestra cómo acceder a un valor de campo conocido en una carpeta especificada, en este caso, el campo Rto en la carpeta jms, que representa el nombre de la cola de respuestas en un mensaje de MQ JMS.

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

Para descubrir el contenido de una MQRFH2 (en contraposición a solicitar campos específicos directamente), puede utilizar el método getFolders que devuelve una lista de MQRFH2.Element, que representa la estructura de una carpeta que podría contener campos y otras carpetas. Estableciendo en nulos un campo o una carpeta, se elimina de la MQRFH2. Cuando se manipula el contenido de la carpeta NameValueData de esta forma, el campo StrucLength se actualiza automáticamente en consecuencia.

Lectura y escritura de secuencias de bytes que no sean objetos MQMessage en IBM MQ classes for Java
Estos ejemplos utilizan clases de cabecera para analizar y manejar el contenido de cabeceras de IBM MQ cuando el origen de datos no es un objeto MQMessage.

Puede utilizar clases de cabecera para analizar y manejar el contenido de cabeceras de IBM MQ incluso cuando el origen de datos no es un objeto MQMessage. La interfaz MQHeader implementada por cada clase de cabecera proporciona los métodos `int read (java.io.DataInput message, int encoding, int characterSet)` y `int write (java.io.DataOutput message, int encoding, int characterSet)`. La clase com.ibm.mq.MQMessage implementa las interfaces java.io.DataInput y java.io.DataOutput. Esto significa que puede utilizar los dos métodos MQHeader para leer y escribir contenido de MQMessage, pasando por alto temporalmente los valores de codificación y CCSID especificados en el descriptor de mensaje. Esto es útil para mensajes que contengan una cadena de cabeceras con distintas codificaciones.

También puede obtener objetos DataInput y DataOutput de otras secuencias de datos, por ejemplo, secuencias de archivos o de socket, o matrices de bytes transportadas en mensajes de JMS. Las clases java.io.DataInputStream implementan DataInput, y las clases java.io.DataOutputStream implementan DataOutput. Este ejemplo lee el contenido de cabeceras de IBM MQ a partir de una matriz de bytes:

```
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);
```

La línea que empieza por MQHeaderIterator se puede sustituir por

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

Este ejemplo graba en una matriz de bytes utilizando DataOutputStream:

```
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 ();
```

Cuando trabaje con secuencias de este modo, tenga cuidado de utilizar los valores correctos para los argumentos de characterSet y de codificación. Cuando lea cabeceras, especifique la codificación y el CCSID con los que se escribió originalmente el contenido de los bytes. Cuando grabe cabeceras, especifique la codificación y el CCSID que desee producir. La conversión de datos la realizan automáticamente las clases de cabeceras.

Creación de clases para nuevos tipos de cabecera en IBM MQ classes for Java
Puede crear clases Java para tipos de cabecera que no se proporcionan con IBM MQ classes for Java.

Para añadir una clase Java que representa un nuevo tipo de cabecera que puede utilizar de la misma forma que cualquier clase de cabecera proporcionada con IBM MQ classes for Java, cree una clase que implemente la interfaz MQHeader. La forma más sencilla de hacerlo es ampliar la clase com.ibm.mq.headers.impl.Header. Este ejemplo produce una clase totalmente funcional que representa la estructura de cabecera MQTM. No es necesario añadir métodos getter y setter individuales para cada campo, pero es útil para usuarios de la clase de cabecera. Los métodos genéricos getValue y setValue que utilizan una serie como nombre de campo serán efectivos para todos los campos definidos en el tipo de cabecera. Los métodos heredados de lectura, escritura y tamaño permitirán leer y escribir instancias del nuevo tipo de cabecera y calcularán correctamente el tamaño de la cabecera basándose en su definición de campo. La definición de tipo se crea una sola vez, aunque se crean varias instancias de esa clase de cabecera. Para que la definición de la nueva cabecera quede disponible para su descodificación utilizando las clases MQ HeaderIterator o MQHeaderList, regístrela utilizando MQHeaderRegistry. Pero observe que, de hecho, la clase de cabecera MQTM ya se proporciona en este paquete y está registrada en el registro predeterminado.

```
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.
}
```

Manejo de mensajes PCF con IBM MQ classes for Java

Se proporcionan clases Java para crear y analizar mensajes estructurados PCF, facilitar el envío de solicitudes PCF y la recopilación de respuestas PCF.

Las clases PCFMessage y MQCFGR representan matrices de estructuras de parámetros PCF. Proporcionan métodos de conveniencia para añadir y recuperar parámetros PCF.

Las estructuras de parámetros PCF se representan mediante las clases MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL y MQCFGR. Estas clases comparten interfaces operativas básicas:

- Métodos para leer y escribir el contenido de mensajes: read (), write () y size ()
- Métodos para manejar parámetros: getValue (), setValue (), getParameter () y otros
- El método enumerador .nextParameter (), que analiza el contenido PCF en un MQMessage

El parámetro de filtro PCF se utiliza en mandatos de consulta para proporcionar una función de filtro. Está encapsulado en las clases siguientes:

- MQCFIF - filtro de enteros
- MQCFSF - filtro de series
- MQCFBF - filtro de bytes

Se proporcionan dos clases agentes, PCFAgent y PCFMessageAgent, para gestionar la conexión con un gestor de colas, la cola del servidor de mandatos y una cola de respuesta asociada. PCFMessageAgent amplía PCFAgent y debe utilizarse normalmente de preferencia a este último. La clase PCFMessageAgent convierte los MQMessages recibidos y los devuelve al interlocutor como una matriz de PCFMessage. PCFAgent devuelve una matriz de MQMessages, que es necesario analizar antes de utilizarla.

Manejo de propiedades de mensaje en IBM MQ classes for Java

Las llamadas de función para procesar descriptores de contexto de mensaje no tienen un equivalente en IBM MQ classes for Java. Para establecer, devolver o suprimir propiedades de descriptores de contexto de mensaje, utilice métodos de la clase MQMessage.

Para obtener información general sobre propiedades de mensaje, consulte “Nombres de propiedades” en la página 28.

En IBM MQ classes for Java, el acceso a los mensajes se realiza mediante la clase MQMessage. Por tanto, no se proporcionan descriptores de contexto de mensaje en el entorno Java y no existe un equivalente a las llamadas de función MQCRTMH, MQDLTMH, MQMHBUF y MQBUFMH de IBM MQ

Para establecer propiedades de descriptor de contexto de mensaje en la interfaz de procedimientos, utilice la llamada MQSETMP. En IBM MQ classes for Java, utilice el método apropiado de la clase MQMessage:

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty

- setDoubleProperty
- setStringProperty
- setObjectProperty

Algunas veces, se hace referencia a estos métodos de forma colectiva como métodos *set*property*.

Para devolver el valor de las propiedades de los descriptores de mensajes en la interfaz de procedimientos, debe utilizar la llamada MQINQMP. En IBM MQ classes for Java, utilice el método apropiado de la clase MQMessage:

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty
- getObjectProperty

Algunas veces, se hace referencia a estos métodos de forma colectiva como métodos *get*property*.

Para suprimir el valor de las propiedades de los descriptores de mensajes en la interfaz de procedimientos, debe utilizar la llamada MQDLTMP. En IBM MQ classes for Java, utilice el método deleteProperty de la clase MQMessage.

Manejo de errores en IBM MQ classes for Java

Manejo de errores procedentes de IBM MQ classes for Java utilizando los bloques de código Java `try` y `catch`.

Los métodos de la interfaz de Java no devuelven un código de terminación y un código de razón. En lugar de ello, emiten una excepción siempre que el código de terminación y el código de razón resultantes de una llamada a IBM MQ sean ambos distintos de cero. Esto simplifica la lógica del programa para que el usuario no necesite comprobar los códigos de retorno después de cada llamada a IBM MQ. Puede decidir en qué puntos del programa desea tratar la posibilidad de anomalía. En los puntos indicados, puede rodear el código con bloques `try` y `catch`, tal como se muestra en el ejemplo siguiente:

```
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() );
}
```

Los códigos de razón de llamada de IBM MQ notificados en excepciones de Java para z/OS se documentan en Códigos de razón y finalización de API.

Las excepciones que se generan mientras se ejecuta una aplicación de IBM MQ classes for Java también se escriben en el archivo de registro. Pero una aplicación puede invocar el método

MQException.logExclude() para impedir que se registren excepciones asociadas a un código de razón determinado. Esto puede ser conveniente hacerlo cuando prevea que se emitirán muchas excepciones asociadas a un código de razón determinado y no desea que el archivo de registro se llene con estas excepciones. Por ejemplo, si la aplicación intenta obtener un mensaje de una cola cada vez que se ejecuta en bucle y, en la mayoría de estos intentos, no espera que haya ningún mensaje adecuado en la cola, puede impedir que se registren las excepciones asociadas al código de razón MQRC_NO_MSG_AVAILABLE. Si una aplicación ha impedido anteriormente que se registren las excepciones asociadas a un código de razón determinado, la aplicación puede ahora permitir que estas excepciones se vuelvan a registrar invocando el método MQException.logInclude().

Algunas veces, el código de razón no proporciona todos los detalles relacionados con el error. Para cada excepción que se emite, es conveniente que la aplicación examine la excepción enlazada. La excepción enlazada puede tener a su vez otra excepción enlazada; de esta forma, las excepciones enlazadas forman una cadena que apunta al problema original subyacente. Una excepción enlazada se implementa utilizando el mecanismo de excepción en cadena de la clase java.lang.Throwable y una aplicación obtiene una excepción enlazada llamando al método Throwable.getCause(). A partir de una excepción que es una instancia de MQException, MQException.getCause() recupera la instancia subyacente de com.ibm.mq.jmqi.JmqiException, y getCause de esta excepción recupera la java.lang.Exception subyacente que ha ocasionado el error.

Obtención y establecimiento de valores de atributo en IBM MQ classes for Java

Para muchos atributos comunes, se proporcionan métodos getXXX() y setXXX(). Otros atributos se pueden acceder utilizando los métodos genéricos inquire() y set().

Para muchos de los atributos comunes, las clases MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess y MQQueueManager contienen métodos getXXX() y setXXX(). Estos métodos permiten obtener y establecer los valores de atributo. Observe que para MQDestination, MQQueue y MQTopic, los métodos son efectivos sólo si especifica los distintivos de inquire y set apropiados cuando abre el objeto.

Para atributos menos comunes, las clases MQQueueManager, MQDestination, MQQueue, MQTopic y MQProcess todas heredan de una clase llamada MQManagedObject. Esta clase define las interfaces inquire() y set().

Cuando crea un nuevo objeto de gestor de colas utilizando el operador *new*, el objeto se abre automáticamente para inquire. Cuando utiliza el método accessProcess() para acceder a un objeto de proceso, el objeto se abre automáticamente para inquire. Cuando utiliza el método accessQueue() para acceder a un objeto de cola, el objeto no se abre automáticamente para las operaciones inquire o set. Esto es así porque la adición de estas opciones puede causar problemas automáticamente con algunos tipos de colas remotas. Para utilizar los métodos inquire, set, getXXX y setXXX en una cola, debe especificar los distintivos inquire y set adecuados en el parámetro openOptions del método accessQueue(). Lo mismo es válido para los objetos de destino y de tema.

Los métodos inquire y set tienen tres parámetros:

- matriz selectors
- matriz intAttrs
- matriz charAttrs

No necesita los parámetros SelectorCount, IntAttrCount y CharAttrLength que se encuentran en MQINQ, porque la longitud de una matriz Java siempre se conoce. El ejemplo siguiente muestra cómo efectuar una consulta sobre una cola:

```
// 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));
```

Programas multihebra en Java

El entorno de ejecución Java es multihebra de forma intrínseca. IBM MQ classes for Java permite que un objeto de gestor de colas sea compartido por varias hebras, pero garantiza que todo el acceso al gestor de colas de destino esté sincronizado.

Los programas multihebra son difíciles de evitar en Java. Considere por ejemplo un programa simple que se conecta a un gestor de colas y abre una cola durante el proceso de inicio. El programa visualiza un solo botón en la pantalla y, cuando el usuario lo pulsa, el programa busca un mensaje en la cola y lo carga.

El entorno de ejecución Java es multihebra de forma intrínseca. Por lo tanto, la inicialización de la aplicación tiene lugar en una sola hebra y el código que se ejecuta en respuesta a la pulsación del botón se ejecuta en una hebra separada (la hebra de la interfaz de usuario).

Con el IBM MQ MQI client basado en el lenguaje C, esto causaría un problema, pues hay limitaciones en el uso compartido de descriptores de contexto por parte de varias hebras. IBM MQ classes for Java relaja esta restricción, permitiendo que un objeto de gestor de colas (y sus objetos asociados de cola, tema y proceso) sean compartidos por varias hebras.

La implementación de IBM MQ classes for Java asegura, para una conexión determinada, (instancia de objeto MQQueueManager), que se sincronice todo el acceso al gestor de colas de IBM MQ. Una hebra que desee emitir una llamada a un gestor de colas queda bloqueada hasta que finalizan todas las demás llamadas en curso para dicha conexión. Si necesita acceso simultáneo al mismo gestor de colas desde varias hebras dentro del programa, cree un nuevo objeto MQQueueManager para cada hebra que necesite acceso simultáneo. (Equivale a emitir una llamada MQCONN independiente para cada hebra).

Nota: No se deben compartir instancias de la clase `com.ibm.mq.MQGetMessageOptions` entre hebras que están solicitando mensajes simultáneamente. Las instancias de esta clase se actualizan con datos durante la solicitud MQGET correspondiente, lo que puede dar lugar a resultados inesperados cuando varias hebras están operando simultáneamente en la misma instancia del objeto.

Utilización de salidas de canal en IBM MQ classes for Java

Visión general de cómo utilizar salidas de canal en una aplicación utilizando las IBM MQ classes for Java.

Los temas siguientes describen cómo escribir una salida de canal en Java, cómo asignarla y cómo pasar datos a ella. A continuación, los temas describen cómo utilizar salidas de canal escritas en C y cómo utilizar una secuencia de salidas de canal.

La aplicación debe tener el permiso de seguridad correcto para cargar la clase de salida de canal.

Creación de una salida de canal en IBM MQ classes for Java

Puede proporcionar sus propias salidas de canal definiendo una clase Java que implemente una interfaz adecuada.

Para implementar una salida, defina una nueva clase Java que implemente la interfaz adecuada. El paquete com.ibm.mq.exits contiene tres interfaces de salida definidas:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Nota: Las salidas de canal sólo se pueden utilizar para conexiones de cliente; no se pueden utilizar para conexiones de enlaces. No puede utilizar una salida de canal Java fuera de IBM MQ classes for Java, por ejemplo si está utilizando una aplicación cliente escrita en C.

Cualquier cifrado TLS definido para una conexión se realiza *después* de haber invocado salidas de emisión y de seguridad. Del mismo modo, el descifrado se realiza *antes* de invocar salidas de recepción y de seguridad.

El ejemplo siguiente define una clase que implementa las tres interfaces:

```
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
  }
}
```

A cada salida se le pasa un objeto MQCXP y un objeto MQCD. Estos objetos representan las estructuras MQCXP y MQCD definidas en la interfaz de procedimientos.

Cualquier clase de salida que escribe debe tener un constructor. Este puede ser el constructor predeterminado o uno que toma un argumento de serie. Si toma una serie, los datos de usuario pasarán a la clase de salida cuando se creen. Si la clase de salida contiene un constructor predeterminado y un constructor de un solo argumento, el constructor de un solo argumento tiene prioridad.

Para las salidas de emisión y de seguridad, el código de salida debe devolver los datos que desea enviar al servidor. Para una salida de recepción, el código de salida debe devolver los datos modificados que desea que IBM MQ interprete.

El cuerpo de salida más simple es:

```
{ return agentBuffer; }
```

No cierre el gestor de colas desde dentro de una salida de canal.

Utilización de clases de salida de canal existentes

En las versiones de IBM MQ anteriores a 7.0, estas salidas se implementan mediante las interfaces MQSendExit, MQReceiveExit y MQSecurityExit, como en el ejemplo siguiente. Este método sigue siendo válido, pero es preferible el nuevo método para obtener una funcionalidad y un rendimiento mejores.

```
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
  }
}
```

Asignación de una salida de canal en IBM MQ classes for Java
Puede asignar una salida de canal utilizando IBM MQ classes for Java.

No hay un equivalente directo del canal de IBM MQ en IBM MQ classes for Java. Las salidas de canal se asignan a un MQQueueManager. Por ejemplo, después de haber definido una clase que implementa la interfaz WMQSecurityExit, una aplicación puede utilizar la salida de seguridad en una de cuatro maneras:

- Asignando una instancia de la clase al campo MQEnvironment.channelSecurityExit antes de crear un objeto MQQueueManager
- Estableciendo como valor del campo MQEnvironment.channelSecurityExit una serie de caracteres que representa la clase de salida de seguridad antes de crear un objeto MQQueueManager
- Creando un par clave/valor en la hashtable de propiedades pasada a MQQueueManager con una clave de CMQC.SECURITY_EXIT_PROPERTY
- Utilizando tabla de definición de canal de cliente (CCDT)

Cualquier salida asignada estableciendo el campo MQEnvironment.channelSecurityExit en una serie de caracteres, creando un par clave/valor en la hashtable de propiedades o utilizando una CCDT, se debe escribir con un constructor predeterminado. Una salida asignada como una instancia de una clase no necesita un constructor predeterminado, dependiendo de la aplicación.

Una aplicación puede utilizar una salida de emisión o recepción de la misma manera. Por ejemplo, el fragmento del código siguiente le muestra cómo utilizar las salidas de seguridad, de emisión y de recepción que se implementan en la clase MyMQExits, que se ha definido anteriormente utilizando MQEnvironment:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Si se utiliza más de un método para asignar una salida de canal, el orden de prioridad es el siguiente:

1. Si el URL de una CCDT se pasa a MQQueueManager, el contenido de la CCDT determina las salidas de canal que se deben utilizar y no se tiene en cuenta cualquier definición de salida especificada en MQEnvironment o en la tabla hash de propiedades.
2. Si no se pasa ningún URL de CCDT, se fusionan las definiciones de salida contenidas en MQEnvironment y la tabla hash
   - Si se ha definido un mismo tipo de salida en MQEnvironment y en la tabla hash, se utiliza la definición contenida en la tabla hash.
   - Si se especifican tipos de salida antiguos y nuevos equivalentes (por ejemplo, el campo sendExit , que sólo se puede utilizar para el tipo de salida utilizado en versiones anteriores a IBM WebSphere MQ 7.0, y el campo de salida channelSend, que se puede utilizar para cualquier salida de emisión), se utiliza la salida nueva (salidachannelSend) en lugar de la salida antigua.

Si ha declarado una salida de canal como una serie, debe permitir que IBM MQ localice el programa de salida de canal. Puede hacerlo de varias maneras, dependiendo del entorno en el que se está ejecutando la aplicación y de cómo se empaquetan los programas de salida de canal.

- Para una aplicación que se ejecuta en un servidor de aplicaciones, debe almacenar los archivos en el directorio que se muestra en Tabla 58 en la página 404 o empaquetados en archivos JAR a los que hace referencia **exitClasspath**.
- Para una aplicación que no se ejecuta en un servidor de aplicaciones, se aplican las normas siguientes:
  - Si las clases de salida de canal están empaquetadas en archivos JAR separados, estos archivos JAR se deben incluir en **exitClasspath**.
  - Si las clases de salida de canal no están empaquetadas en archivos JAR, los archivos de clase se pueden almacenar en el directorio que se muestra en Tabla 58 en la página 404 o en cualquier directorio de la vía de acceso de clases del sistema JVM o **exitClasspath**.

La propiedad **exitClasspath** se puede especificar de cuatro maneras. En orden de prioridad, son las siguientes:

1. La propiedad del sistema com.ibm.mq.exitClasspath (definida en la línea de mandatos utilizando la opción `-D`)
2. La stanza exitPath del archivo `mqclient.ini`
3. Una entrada de tabla hash con la clave CMQC.EXIT_CLASSPATH_PROPERTY
4. La variable MQEnvironment **exitClasspath**

Para separar vías de acceso, utilice el carácter especificado por java.io.File.pathSeparator

Tabla 58. Directorio para los programas de salida de canal

Plataforma	**Directorio**
AIX AIX Linux Linux	`/var/mqm/exits` (programas de salida de canal de 32 bits) `/var/mqm/exits64` (programas de salida de canal de 64 bits)
Windows	*dir_datos_instalación*\exits

Nota: *dir_datos_instalación* es el directorio que ha elegido para los archivos de datos de IBM MQ durante la instalación. El directorio predeterminado es `C:\ProgramData\IBM\MQ`.

Transferencia de datos a salidas de canal en IBM MQ classes for Java
Puede pasar datos a salidas de canal y devolver datos desde salidas de canal a la aplicación.

El parámetro agentBuffer

Para una salida de emisión, el parámetro *agentBuffer* contiene los datos que se van a enviar. Para una salida de recepción o seguridad, el parámetro *agentBuffer* contiene los datos que se acaban de recibir. No es necesario un parámetro de longitud, pues la expresión agentBuffer.limit() indica la longitud de la matriz.

Para las salidas de emisión y de seguridad, el código de salida debe devolver los datos que desea enviar al servidor. Para una salida de recepción, el código de salida debe devolver los datos modificados que desea que IBM MQ interprete.

El cuerpo de salida más simple es:

```
{ return agentBuffer; }
```

Las salidas de canal se invocan con un almacenamiento intermedio que tiene una matriz de seguridad. Para obtener el mejor rendimiento, la salida debe devolver un almacenamiento intermedio con una matriz auxiliar.

Datos de usuario

Si una aplicación se conecta a un gestor de colas estableciendo channelSecurityExit, channelSendExit o channelReceiveExit, se pueden transferir 32 bytes de datos de usuario a la clase de salida de canal adecuada cuando esta se invoca, utilizando los campos channelSecurityExitUserData, channelSendExitUserData o channelReceiveExitUserData. Estos datos de usuario están disponibles en la clase de salida de canal, pero se renueva cada vez que se invoca la salida. Por consiguiente, los cambios efectuados en los datos de usuario en la salida de canal se perderán. Si desea hacer cambios persistentes en los datos en una salida de canal, utilice la MQCXP exitUserArea. Los datos en este campo se mantienen entre invocaciones de la salida.

Si la aplicación establece securityExit, sendExit o receiveExit, no se pueden transferir datos de usuario a estas clases de salida de canal.

Si una aplicación utiliza una tabla de definición de canal de cliente (CCDT) para conectarse a un gestor de colas, los datos de usuario especificados en una definición de canal de conexión de cliente se pasan a las clases de salida de canal cuando se invocan. Si desea más información sobre la utilización de las tablas de definición de canal de cliente, consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for Java" en la página 387.

Utilización de salidas de canal no escritas en Java con IBM MQ classes for Java
Cómo utilizar programas de salida de canal escritos en C desde una aplicación de Java.

En IBM MQ, puede especificar el nombre de un programa de salida de canal escrito en C como una serie pasada a los campos channelSecurityExit, channelSendExit o channelReceiveExit en el objeto MQEnvironment o las propiedades Hashtable. Pero no puede utilizar una salida de canal escrita en Java en una aplicación escrita en otro lenguaje.

Especifique el nombre del programa de salida en el formato `library(function)` y asegúrese de que la ubicación del programa de salida esté especificada tal como se describe en Vías de acceso de salidas.

Para obtener información sobre cómo escribir una salida de canal en C, consulte "Programas de salida de canal para canales de mensajes" en la página 986.

Utilización de una secuencia de salidas de canal de emisión o recepción en IBM MQ classes for Java
Una aplicación de IBM MQ classes for Java puede utilizar una secuencia de salidas de canal de emisión o recepción que se ejecutan sucesivamente.

Para utilizar una secuencia de salidas de emisión, puede crear un objeto List o String que contiene las salidas de emisión. Si se utiliza un objeto List, cada elemento de la lista puede ser uno de los siguientes elementos:

- Una instancia de una clase definida por el usuario que implementa la interfaz WMQSendExit
- Una instancia de una clase definida por el usuario que implementa la interfaz MQSendExit (para una salida de emisión escrita en Java)
- Una instancia de la clase MQExternalSendExit (para una salida de emisión no escrita en Java)
- Una instancia de la clase MQSendExitChain
- Una instancia de la clase String

Una lista no puede contener otra lista.

La aplicación puede utilizar una secuencia de salidas de recepción de una manera similar.

Si se utiliza una serie, debe constar de una o más definiciones de salida separadas por comas, cada una de las cuales puede ser el nombre de una clase Java o un programa C con el formato `library(function)`.

A continuación, la aplicación asigna el objeto List o String al campo MQEnvironment.channelSendExit antes de crear un objeto MQQueueManager.

El contexto de información que se pasó a las salidas está únicamente dentro del dominio de las salidas. Por ejemplo, si una salida Java y una salida C están encadenadas, la presencia de la salida Java no tiene ningún efecto sobre la salida C.

Utilización de clases de cadena de salidas

En versiones anteriores a IBM WebSphere MQ 7.0, se proporcionaban dos clases para permitir secuencias de salidas:

- MQSendExitChain, que implementa la interfaz MQSendExit
- MQReceiveExitChain, que implementa la interfaz MQReceiveExit

El uso de estas clases sigue siendo válido, pero es preferible el nuevo método. La utilización de las interfaces de IBM MQ Classes for Java significa que la aplicación sigue dependiendo de `com.ibm.mq.jar`. Si se utiliza el nuevo conjunto de interfaces contenidas en el paquete `com.ibm.mq.exits`, no hay ninguna dependencia respecto de `com.ibm.mq.jar`.

Para utilizar una secuencia de salidas de emisión, una aplicación ha creado una lista de objetos, donde cada objeto era uno de estos elementos:

- Una instancia de una clase definida por el usuario que implementa la interfaz MQSendExit (para una salida de emisión escrita en Java)
- Una instancia de la clase MQExternalSendExit (para una salida de emisión no escrita en Java)
- Una instancia de la clase MQSendExitChain

La aplicación ha creado un objeto MQSendExitChain pasando esta lista de objetos como parámetro en el constructor. A continuación, la aplicación habría asignado el objeto MQSendExitChain al campo MQEnvironment.sendExit antes de crear un objeto MQQueueManager.

Compresión de canal en IBM MQ classes for Java

La compresión de los datos que fluyen por un canal puede mejorar el rendimiento del canal y reducir el tráfico de la red. IBM MQ classes for Java utilizan la función de compresión incorporada en IBM MQ.

Esta función proporcionada con IBM MQ le permite comprimir los datos que circulan por los canales de mensajes y canales MQI. En ambos tipos de canal puede comprimir datos de cabecera y datos de mensaje por separado. De forma predeterminada, no se comprime ningún dato en un canal. Para obtener una descripción completa de la compresión de canal, incluida la forma en que se implementa en IBM MQ, consulte Compresión de datos (COMPMSG)y Compresión de cabecera (COMPHDR).

Una aplicación de IBM MQ classes for Java especifica las técnicas que se pueden utilizar para comprimir datos de cabecera o de mensaje en una conexión de cliente creando un objeto java.util.Collection. Cada técnica de compresión es un objeto Integer de la colección, y el orden en el que la aplicación añade las técnicas de compresión a la colección es el orden en que se negocian las técnicas de compresión con el gestor de colas cuando se inicia la conexión de cliente. A continuación, la aplicación puede asignar la colección al campo hdrCompList, para los datos de cabecera, o al campo msgCompList, para los datos de mensaje, en la clase MQEnvironment. Cuando la aplicación está preparada, puede iniciar la conexión de cliente creando un objeto MQQueueManager.

Los fragmentos de código siguientes muestran el método descrito. El primer fragmento de código muestra cómo implementar la compresión de datos de cabecera:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

El segundo fragmento de código muestra cómo implementar la compresión de datos de mensaje:

```
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);
```

En el segundo ejemplo, las técnicas de compresión se negocian en el orden RLE, a continuación ZLIBHIGH, cuando se inicia la conexión del cliente. La técnica de compresión que se selecciona no se puede modificar durante la duración del objeto MQQueueManager.

Las técnicas de compresión para datos de cabecera y de mensaje soportadas por el cliente y el gestor de colas en una conexión de cliente se pasan a una salida de canal como colecciones en los campos hdrCompList y msgCompList de un objeto MQChannelDefinition. Las técnicas reales que se utilizan actualmente para comprimir los datos de cabecera y de mensaje en una conexión de cliente se pasan a una salida de canal en los campos CurHdrCompression y CurMsgCompression de un objeto MQChannelExit.

Si la compresión se utiliza en una conexión de cliente, los datos se comprimen antes de procesar cualquier salida de emisión de canal y los datos se extraen después de procesar las salidas de recepción de canal. Por ello, los datos pasados a salidas de emisión y recepción están en un estado comprimido.

Si desea obtener más información sobre cómo especificar técnicas de compresión y sobre qué técnicas de compresión hay disponibles, consulte Clase com.ibm.mq.MQEnvironment y Interfaz com.ibm.mq.MQC .

Compartir una conexión TCP/IP en IBM MQ classes for Java

Se pueden crear varias instancias de un canal MQI para que compartan una sola conexión TCP/IP.

En IBM MQ classes for Java, utilice la variable MQEnvironment.sharingConversations para controlar el número de conversaciones que pueden compartir una misma conexión TCP/IP.

El atributo SHARECNV es un enfoque sin garantías para la compartición de conexiones. Por lo tanto, cuando se utiliza un valor SHARECNV mayor que 0 con IBM MQ classes for Java, no es seguro que una nueva solicitud de conexión compartirá siempre una conexión ya establecida.

Agrupación de conexiones en IBM MQ classes for Java

IBM MQ classes for Java permite que las conexiones libres se agrupen para su reutilización.

IBM MQ classes for Java proporciona soporte adicional para las aplicaciones que gestionan varias conexiones con gestores de colas de IBM MQ. Cuando una conexión ya no es necesaria, en lugar de eliminarla, se puede añadir a una agrupación de conexiones y volver a utilizarla más tarde. Esto puede proporcionar una mejora notable del rendimiento para las aplicaciones y el middleware que se conectan en serie a gestores de cola arbitrarios.

IBM MQ proporciona una agrupación de conexiones predeterminada. Las aplicaciones pueden activar o desactivar esta agrupación de conexiones registrando o desregistrando señales a través de la clase MQEnvironment. Si la agrupación está activa cuando IBM MQ classes for Java construye un objeto MQQueueManager, éste busca en esta agrupación predeterminada y reutiliza cualquier conexión adecuada. Cuando se produce una llamada MQQueueManager.disconnect(), la conexión subyacente se devuelve a la agrupación.

Como alternativa, las aplicaciones pueden construir una agrupación de conexiones MQSimpleConnectionManager para una finalidad determinada. Entonces, la aplicación puede especificar esa agrupación durante la construcción de un objeto MQQueueManager o pasar esa agrupación a MQEnvironment para utilizarla como agrupación de conexiones predeterminada.

Para impedir que las conexiones utilicen demasiados recursos, puede limitar el número total de conexiones que un objeto MQSimpleConnectionManager puede manejar y puede limitar el tamaño de la agrupación de conexiones. El establecimiento de límites es útil si existen demandas conflictivas de conexiones dentro de una JVM.

De forma predeterminada, el método getMaxConnections() devuelve el valor cero, lo que significa que no existe ningún límite para el número de conexiones que el objeto MQSimpleConnectionManager puede manejar. Puede establecer un límite utilizando el método setMaxConnections(). Si establece un límite y éste se alcanza, una petición de conexión adicional puede hacer que se emita una MQException con un código de razón MQRC_MAX_CONNS_LIMIT_REACHED.

Control de la agrupación de conexiones predeterminada en IBM MQ classes for Java
Este ejemplo muestra cómo utilizar la agrupación de conexiones predeterminada.

Considere la aplicación de ejemplo siguiente, 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 toma una lista de gestores de cola locales de la línea de mandatos, se conecta a cada uno, uno después de otro, y realiza alguna operación. Sin embargo, cuando la línea de mandatos enumera el mismo gestor de colas varias veces, resulta más eficaz conectarse sólo una vez y reutilizar dicha conexión varias veces.

IBM MQ classes for Java proporciona una agrupación de conexiones predeterminada que puede utilizar para realizar esta función. Para habilitar la agrupación, utilice uno de los métodos MQEnvironment.addConnectionPoolToken(), y para inhabilitarla, utilice MQEnvironment.removeConnectionPoolToken().

Funcionalmente, la aplicación de ejemplo siguiente, MQApp2, es idéntica a MQApp1, pero sólo se conecta una vez a cada gestor de colas.

```
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);

      }
}
```

La primera línea en negrita activa la agrupación de conexiones predeterminada, al registrar un objeto MQPoolToken con MQEnvironment.

El constructor MQQueueManager ahora busca en la agrupación una conexión adecuada y sólo crea una conexión para el gestor de colas si no encuentra ninguna existente. La llamada qmgr.disconnect() devuelve la conexión a la agrupación de conexiones para reutilizarla más tarde. Estas llamadas de API son las mismas que las de la aplicación de ejemplo MQApp1.

La segunda línea resaltada desactiva la agrupación de conexiones predeterminada, lo cual destruye todas las conexiones del gestor de colas almacenadas en la agrupación. Esto es importante, pues en otro caso la aplicación podría terminar con varias conexiones de gestor de colas activas en la agrupación. Esta situación podría producir errores que aparecerían en los archivos de registro del gestor de colas.

Cuando una aplicación utiliza una tabla de definición de canal de cliente (CCDT) para conectarse a un gestor de colas, el constructor MQQueueManager primero busca en la tabla una definición de canal de conexión de cliente adecuada. Si encuentra una, el constructor busca en la agrupación de conexiones

predeterminada una conexión que se pueda utilizar para el canal. Si el constructor no puede encontrar una conexión adecuada en la agrupación, busca en la tabla de definición de canal de cliente la siguiente definición adecuada y continúa tal como se ha descrito anteriormente. Si el constructor completa la búsqueda de la tabla de definición de canal de cliente y no ha encontrado ninguna conexión adecuada en la agrupación, el constructor inicia una segunda búsqueda de la tabla. Durante esta búsqueda, el constructor intenta crear una nueva conexión para cada definición de canal de conexión con el cliente adecuada y utiliza la primera conexión que logra crear.

La agrupación de conexiones predeterminada almacena diez conexiones no utilizadas como máximo y mantiene activas las conexiones no utilizadas durante un máximo de cinco minutos. La aplicación puede modificar estos valores (para obtener información más detallada, consulte el apartado “Suministro de una agrupación de conexiones distinta en IBM MQ classes for Java” en la página 410).

En vez de utilizar MQEnvironment para suministrar una señal MQPoolToken, la aplicación puede construir la suya propia:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Algunos proveedores de aplicaciones o middleware facilitan subclases de MQPoolToken para pasar información a una agrupación de conexiones personalizada. Se pueden construir y pasar a addConnectionPoolToken(), de modo que se pueda pasar información adicional a la agrupación de conexiones.

La agrupación de conexiones predeterminada y varios componentes en IBM MQ classes for Java
Este ejemplo muestra cómo añadir o eliminar MQPoolTokens de un grupo estático de objetos MQPoolToken registrados.

MQEnvironment mantiene un conjunto de objetos MQPoolToken registrados. Para añadir o eliminar MQPoolTokens del conjunto, utilice los métodos siguientes:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Una aplicación puede constar de varios componentes que existan de modo independiente y trabajen utilizando un gestor de colas. En este tipo de aplicación, cada componente debe añadir una MQPoolToken al MQEnvironment establecido para toda su duración.

Por ejemplo, la aplicación de ejemplo MQApp3 crea diez hebras e inicia cada una de ellas. Cada hebra registra su MQPoolToken específico, espera un tiempo determinado y, a continuación, se conecta al gestor de colas. Después la hebra se desconecta y elimina su MQPoolToken.

La agrupación de conexiones predeterminada permanece activa mientras hay, como mínimo, una señal en el conjunto de MQPoolTokens, por lo que permanece activa mientras dura la aplicación. La aplicación no necesita mantener un objeto maestro para el control global de las hebras.

```
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);
      }
}
```

Suministro de una agrupación de conexiones distinta en IBM MQ classes for Java
Este ejemplo muestra cómo utilizar la clase **com.ibm.mq.MQSimpleConnectionManager** para suministrar una agrupación de conexiones diferente.

Esta clase proporciona recursos básicos para la agrupación de conexiones y las aplicaciones pueden utilizar esta clase para personalizar el comportamiento de la agrupación.

Después de crear una instancia de MQSimpleConnectionManager, se puede especificar un en el constructor MQQueueManager. A continuación, MQSimpleConnectionManager gestiona la conexión subyacente del MQQueueManager construido. Si MQSimpleConnectionManager contiene una conexión adecuada de la agrupación, esa conexión se vuelve a utilizar y se devuelve a MQSimpleConnectionManager después de una llamada a MQQueueManager.disconnect().

El fragmento de código siguiente muestra este comportamiento:

```
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);
```

La conexión que se crea durante el primer constructor MQQueueManager se almacena en myConnMan después de la llamada a qmgr.disconnect(). A continuación, la conexión se vuelve a utilizar durante la segunda llamada al constructor MQQueueManager.

La segunda línea habilita MQSimpleConnectionManager. La última línea inhabilita MQSimpleConnectionManager, eliminando todas las conexiones mantenidas en la agrupación. MQSimpleConnectionManager está, de forma predeterminada, en MODE_AUTO, que se describe más adelante en este apartado.

MQSimpleConnectionManager asigna conexiones según la utilización más reciente y destruye las conexiones conforme a la utilización menos reciente. De forma predeterminada, una conexión se destruye si no se ha utilizado durante cinco minutos, o si hay más de diez conexiones no utilizadas en la agrupación. Puede modificar estos valores invocando MQSimpleConnectionManager.setTimeout().

También puede establecer un MQSimpleConnectionManager para utilizarlo como agrupación de conexiones predeterminada cuando no se proporciona ningún gestor de conexiones en el constructor MQQueueManager.

Esto se muestra en la aplicación siguiente:

```
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);
      }
}
```

Las líneas en negrita crean y configuran un objeto MQSimpleConnectionManager. La configuración hace lo siguiente:

- Finaliza conexiones que no se han utilizado durante una hora
- Limita el número de conexiones gestionadas por myConnMan a 75
- Limita el número de conexiones no utilizadas de la agrupación a 50
- Establece MODE_AUTO, que es el valor predeterminado. Esto significa que la agrupación sólo está activa si es el gestor de conexiones predeterminado y existe como mínimo un token en el conjunto de MQPoolTokens mantenidos por MQEnvironment.

A continuación, el nuevo MQSimpleConnectionManager se establece como gestor de conexiones predeterminado.

En la última línea, la aplicación llama a MQApp3.main(). Esto ejecuta un número de hebras, donde cada hebra utiliza IBM MQ de forma independiente. Estas hebras utilizan myConnMan cuando crean conexiones.

Coordinación de JTA/JDBC utilizando IBM MQ classes for Java

IBM MQ classes for Java soporta el método MQQueueManager.begin (), que permite que IBM MQ actúe como coordinador de una base de datos que proporciona un controlador compatible JDBC de tipo 2 o JDBC de tipo 4.

Este soporte no está disponible en todas las plataformas. Para comprobar qué plataformas soportan la coordinación de JDBC, consulte Requisitos del sistema para IBM MQ.

Para utilizar el soporte de XA-JTA, debe utilizar la biblioteca de conmutadores especial de JTA. El método para utilizar esta biblioteca varía dependiendo de si está utilizando Windows o una de las demás plataformas.

Configuración de la coordinación JTA/JDBC en Windows

La biblioteca XA se suministra como una DLL cuyo nombre tiene el formato `jdbcxxx.dll`.

La `jdbcora12.dll` proporciona compatibilidad con Oracle 12C, para una instalación de servidor IBM MQ para Windows.

En los sistemas Windows, la biblioteca XA se proporciona como una DLL completa. El nombre de esta DLL es `jdbcxxx.dll`, donde `xxx` indica la base de datos para la que se ha compilado la biblioteca de conmutadores. Esta biblioteca reside en el directorio `java\lib\jdbc` o `java\lib64\jdbc` de la instalación de IBM MQ classes for Java. Debe declarar la biblioteca XA, también descrita como archivo de carga conmutada, en el gestor de colas. Utilizar IBM MQ Explorer. Especifique los detalles del archivo de carga conmutada en el panel de propiedades de gestor de colas, bajo el gestor de recursos XA. Especifique únicamente el nombre de la biblioteca. Por ejemplo:

Para una base de datos Db2, establezca el campo SwitchFile en: `dbcdb2`

Para una base de datos Oracle, establezca el campo SwitchFile en: `jdbcora`

Notas:

1. Oracle 12C está soportado por IBM MQ classes for Java, solo en IBM MQ para Windows.
2. La versión soportada de Oracle 12C es 12.1.0.1.0 Enterprise Edition y los fixpacks futuros.
3. Las bases de datos Oracle de 64-bits en Windows de 64-bits requieren el cliente Oracle de 32-bits.
4. Utilizando IBM MQ classes for Java, IBM MQ puede actuar como coordinador de transacción. Sin embargo, no es posible participar en una transacción de estilo JTA.

Configuración de la coordinación JTA/JDBC en plataformas distintas de Windows
Se suministran archivos de objeto. Enlace el archivo de objeto adecuado utilizando el archivo make proporcionado y declare el archivo en el gestor de colas utilizando el archivo de configuración.

Para cada sistema de gestión de bases de datos, IBM MQ proporciona dos archivos de objeto. Es necesario enlazar un archivo de objeto para crear una biblioteca de conmutadores de 32 bits y enlazar el otro archivo de objeto para crear una biblioteca de conmutadores de 64 bits. Para Db2, el nombre de cada archivo de objeto es `jdbcdb2.o`, y para Oracle el nombre de cada archivo de objeto es `jdbcora.o`.

Debe enlazar cada archivo de objeto utilizando el archivo make adecuado que se proporciona con IBM MQ. Una biblioteca de conmutadores necesita otras bibliotecas, que pueden estar almacenadas en diversas ubicaciones en sistemas diferentes. Sin embargo, una biblioteca de conmutación no puede utilizar la variable de entorno de vía de acceso de biblioteca para localizar estas bibliotecas porque el gestor de colas carga la biblioteca de conmutación, que se ejecuta en un entorno `setuid` . Por lo tanto, el archivo make proporcionado garantiza que una biblioteca de conmutadores contenga las vías de acceso completas de estas bibliotecas.

Para crear una biblioteca de conmutadores, emita un mandato **make** con el formato siguiente. Para crear una biblioteca de conmutadores de 32 bits, especifique el mandato en el directorio `/java/lib/jdbc` de la instalación de IBM MQ . Para crear una biblioteca de conmutadores de 64 bits, especifique el mandato en el directorio `/java/lib64/jdbc` .

```
make DBMS
```

donde *DBMS* es el sistema de gestión de bases de datos para el que está creando la biblioteca de conmutadores. Los valores válidos son db2 para Db2 y `oracle` para Oracle.

Nota:

- Para ejecutar aplicaciones de 32 bits, debe crear una biblioteca de conmutadores de 32 bits y una biblioteca de conmutadores de 64 bits para cada sistema de gestión de bases de datos que esté utilizando. Para ejecutar aplicaciones de 64 bits, sólo necesita crear una biblioteca de conmutadores de 64 bits. Para Db2, el nombre de cada biblioteca de conmutadores es `jdbcdb2` y, para Oracle, el nombre de cada biblioteca de conmutadores es `jdbcora`. Los archivos make garantizan que las bibliotecas de conmutadores de 32 y 64 bits se almacenan en directorios diferentes de IBM MQ. Una biblioteca de conmutadores de 32 bits se almacena en el directorio `/java/lib/jdbc` y una biblioteca de conmutadores de 64 bits se almacena en el directorio `/java/lib64/jdbc` .
- Puesto que puede instalar Oracle en cualquier lugar de un sistema, los archivos make utilizan la variable de entorno **ORACLE_HOME** para localizar dónde está instalado Oracle .
- Si IBM MQ está instalado en una ubicación distinta de la ubicación predeterminada, modifique el valor de **MQ_INSTALLATION_PATH** en el archivo make.

Después de haber creado las bibliotecas de conmutación para Db2, Oracleo ambos, debe declararlas en el gestor de colas. Si el archivo de configuración del gestor de colas (qm.ini) ya contiene stanzas `XAResourceManager` para bases de datos Db2 o Oracle , debe sustituir la entrada `SwitchFile` en cada stanza por una de las siguientes:

Para una base de datos Db2

```
SwitchFile=jdbcdb2
```

Para una base de datos Oracle

```
SwitchFile=jdbcora
```

No especifique la vía de acceso completa de la biblioteca de conmutadores de 32 bits o 64 bits. Especifique únicamente el nombre de la biblioteca.

Si el archivo de configuración del gestor de colas todavía no contiene stanzas `XAResourceManager` para bases de datos Db2 o Oracle , o si desea añadir stanzas `XAResourceManager` adicionales, consulte Administración IBM MQ para obtener información sobre cómo construir una stanza `XAResourceManager` . Sin embargo, cada entrada `SwitchFile` de una nueva stanza `XAResourceManager` debe ser exactamente como se ha descrito anteriormente para una base de datos Db2 o Oracle . También debe incluir la entrada `ThreadOfControl=PROCESS`.

Una vez que haya actualizado el archivo de configuración del gestor de colas, y que haya comprobado que están establecidas las variables de entorno de base de datos adecuadas, puede reiniciar el gestor de colas.

Utilización de la coordinación JTA/JDBC
Codifique las llamadas de API tal como se muestra en el ejemplo suministrado.

La secuencia básica de las llamadas de API para una aplicación de usuario es la siguiente:

```
  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()
```

El parámetro `xads` de la llamada getJDBCConnection es la implementación específica de base de datos de la interfaz XADataSource que define los detalles de la base de datos a la que debe realizarse la conexión. Consulte la documentación de su base de datos para determinar cómo crear un objeto XADataSource adecuado para pasarlo a getJDBCConnection.

Debe también actualizar la vía de acceso de clases con los archivos JAR adecuados específicos de la base de datos para realizar tareas de JDBC.

Si se debe conectar a varias bases de datos, debe llamar a getJDBCConnection varias veces para realizar la transacción a través de varias conexiones.

Existen dos modalidades de getJDBCConnection, que son un reflejo de las dos modalidades de XADataSource.getXAConnection:

```
  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
```

Estos métodos declaran Exception en sus cláusulas throws para evitar problemas con el verificador JVM en los clientes que no utilizan las funciones de JTA. La excepción real emitida es javax.transaction.xa.XAException, que requiere que el archivo jta.jar se añada a la vía de acceso de clases para los programas que no lo necesitaban anteriormente.

Para utilizar el soporte de JTA/JDBC, debe incluir la sentencia siguiente en la aplicación:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Limitaciones y problemas conocidos con la coordinación de JTA/JDBC
Algunos de los problemas y las limitaciones del soporte JTA/JDBC dependen del sistema de gestión de bases de datos en uso, por ejemplo, los controladores JDBC probados se comportan de forma distinta cuando la base de datos se cierra mientras se ejecuta una aplicación. Si la conexión a la base de datos que utiliza una aplicación está rota, la aplicación puede realizar pasos para restablecer una nueva conexión con el gestor de colas y la base de datos de modo que pueda utilizar esas nuevas conexiones para realizar el trabajo transaccional necesario.

Debido a que el soporte de JTA/JDBC realiza llamadas a controladores JDBC, la implementación de estos controladores JDBC puede tener un efecto significativo en el comportamiento del sistema. En especial, los controladores JDBC que se han probado se presentan de modo diferente cuando se apaga la base de datos mientras una aplicación está en ejecución.

Importante: Evite siempre cerrar repentinamente una base de datos mientras hay aplicaciones que están manteniendo conexiones abiertas con la base de datos.

Nota: Una aplicación de IBM MQ classes for Java se debe conectar utilizando la modalidad de enlaces para hacer que IBM MQ actúe como coordinador de base de datos.

Varias stanzas XAResourceManager
El uso de más de una stanza XAResourceManager en un archivo de configuración del gestor de colas, `qm.ini`, no está soportado. Solo se tiene en cuenta la primera stanza XAResourceManager especificada.

Db2

Algunas veces, Db2 devuelve un error SQL0805N. Este problema se puede resolver con el mandato de CLP siguiente:

```
DB2 bind @db2cli.lst blocking all grant public
```

Para obtener más información, consulte la documentación de Db2.

La stanza XAResourceManager se debe configurar para utilizar ThreadOfControl=PROCESS. Para Db2 8.1 y superior, esto no coincide con la hebra predeterminada del valor de control para Db2, por lo que se debe especificar toc=p en la serie de apertura XA. A continuación se muestra un ejemplo de stanza XAResourceManager para Db2 con coordinación JTA/JDBC:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Esto no impide que las aplicaciones de Java que utilizan la coordinación JTA/JDBC sean aplicaciones multihebra.

Oracle
Invocar el método Connection.close() JDBC después de MQQueueManager.disconnect() genera una SQLException. Invoque Connection.close() antes de invocar MQQueueManager.disconnect() u omita la llamada a Connection.close().

Manejo de problemas con las conexiones de base de datos

Cuando una aplicación de IBM MQ classes for Java utiliza el soporte de JTA/JDBC que es proporcionado por IBM MQ, normalmente ejecuta los pasos siguientes:

1. Crea un nuevo objeto MQQueueManager para representar una conexión con el gestor de colas que actuará como gestor de transacciones.
2. Construye un objeto XADataSource que contiene detalles acerca de cómo conectar con la base de datos que se incluirá en la transacción.

3. Llama al método MQQueueManager.getJDBCConnection (XADataSource) pasando el XADataSource que se ha creado previamente. Esto hace que las IBM MQ classes for Java establezcan una conexión con la base de datos.
4. Llama al método MQQueueManager.begin() para iniciar la transacción XA.
5. Realiza el trabajo de mensajería y base de datos.
6. Cuando se ha completado todo el trabajo necesario, llama al método MQQueueManager.commit (). Esto completa la transacción XA.
7. Si es necesaria una transacción XA nueva en este punto, la aplicación puede repetir los pasos 4, 5 y 6.
8. Cuando la aplicación ha finalizado, debe cerrar la conexión de base de datos que se ha creado en el paso 3 y luego llamar al método MQQueueManager.disconnect() para desconectarse del gestor de colas.

Las IBM MQ classes for Java mantienen una lista interna de todas las conexiones de base de datos que se han creado cuando una aplicación llama a MQQueueManager.getJDBCConnection(XADataSource). Si un gestor de colas necesita comunicarse con la base de datos durante el proceso de la transacción XA, se lleva a cabo el siguiente proceso:

1. El gestor de colas llama a las IBM MQ classes for Java, pasando detalles de la llamada XA que se debe pasar a la base de datos.
2. A continuación, las IBM MQ classes for Java buscan la conexión adecuada en la lista y utilizan esa conexión para encauzar la llamada XA hacia la base de datos.

Si la conexión con la base de datos se pierde en cualquier punto durante este proceso, la aplicación debe:

1. Restituir cualquier trabajo existente que se haya realizado bajo la transacción, invocando el método MQQueueManager.backout().
2. Cerrar la conexión de base de datos. Esto debe hacer que las IBM MQ classes for Java eliminen de su lista interna los detalles de la conexión de base de datos rota.
3. Desconectar del gestor de colas, llamando al método MQQueueManager.disconnect().
4. Establecer una nueva conexión con el gestor de colas, mediante la construcción de un nuevo objeto MQQueueManager.
5. Crear una nueva conexión de base de datos, llamando al método MQQueueManager.getJDBCConnection(XADataSource).
6. Realizar de nuevo el trabajo transaccional.

Esto permite que la aplicación restablezca una nueva conexión con el gestor de colas y la base de datos y luego utilice esas conexiones para realizar el trabajo transaccional necesario.

Soporte de Transport Layer Security (TLS) en IBM MQ classes for Java

Las aplicaciones cliente de IBM MQ classes for Java son compatibles con el cifrado TLS. Es necesario que un proveedor de JSSE utilice el cifrado TLS.

Las aplicaciones cliente de IBM MQ classes for Java que utilizan TRANSPORT(CLIENT) dan soporte al cifrado TLS. TLS proporciona cifrado de la comunicación, autenticación e integridad de los mensajes. Se suele utilizar para proteger las comunicaciones entre dos interlocutores cualesquiera en Internet o dentro de una intranet.

IBM MQ classes for Java utiliza Java Secure Socket Extension (JSSE) para gestionar el cifrado TLS y por tanto necesita un proveedor JSSE. Las JVM de JSE v1.4 tienen proveedor JSSE incorporado. Los detalles acerca de la gestión y almacenamiento de certificados pueden variar en función del proveedor. Si desea obtener más información sobre este tema, consulte la documentación de su proveedor JSSE.

En este apartado se da por supuesto que el proveedor JSSE está instalado y configurado correctamente, y que se han instalado los certificados pertinentes y se han puesto a la disposición del proveedor JSSE.

Si la aplicación cliente de IBM MQ classes for Java utiliza una tabla de definición de canal de cliente (CCDT) para conectar con un gestor de colas, consulte “Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for Java” en la página 387.

Habilitación de TLS en IBM MQ classes for Java
Para habilitar TLS, especifique una CipherSuite. Hay dos maneras de especificar una CipherSuite.

TLS sólo está soportado para conexiones de cliente. Para habilitar TLS, debe especificar la CipherSuite que se debe utilizar en la comunicación con el gestor de colas, y esta CipherSuite debe coincidir con la CipherSpec establecida en el canal de destino. Además, el proveedor JSSE debe ser compatible con la CipherSuite especificada. Sin embargo, las CipherSuites son distintas de las CipherSpecs y, por tanto, tienen nombres distintos. La sección "CipherSpecs y CipherSuites de TLS en IBM MQ classes for Java" en la página 420 contiene una tabla que correlaciona las CipherSpecs soportadas por IBM MQ con las CipherSuites equivalentes tal como las conoce JSSE.

Para habilitar TLS, especifique la CipherSuite utilizando la variable de miembro estático sslCipherSuite de MQEnvironment. El ejemplo siguiente establece conexión con un canal SVRCONN denominado SECURE.SVRCONN.CHANNEL, que se ha configurado para utilizar TLS con una CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA256:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Aunque la CipherSpec del canal es TLS_RSA_WITH_AES_128_CBC_SHA256, la aplicación de Java debe especificar la CipherSuite SSL_RSA_WITH_AES_128_CBC_SHA256. Consulte el "CipherSpecs y CipherSuites de TLS en IBM MQ classes for Java" en la página 420 para ver una lista de las correlaciones entre las CipherSpecs y las CipherSuites.

Una aplicación puede también especificar una CipherSuite estableciendo la propiedad de entorno CMQC.SSL_CIPHER_SUITE_PROPERTY.

Como alternativa, utilice la tabla de definición de canal de cliente (CCDT). Para obtener más información, consulte "Utilización de una tabla de definiciones de canal de cliente con IBM MQ classes for Java" en la página 387

Si requiere que una conexión de cliente utiliza una CipherSuite soportada por el proveedor FIPS de IBM Java JSSE (IBMJSSEFIPS), una aplicación puede establecer el campo sslFipsRequired de la clase MQEnvironment en `true`. Como alternativa, la aplicación puede establecer la propiedad de entorno CMQC.SSL_FIPS_REQUIRED_PROPERTY. El valor predeterminado es `false`, lo que significa que una conexión de cliente puede utilizar cualquier CipherSuite que sea compatible con IBM MQ.

Si una aplicación utiliza más de una conexión de cliente, el valor del campo sslFipsRequired que se utiliza cuando la aplicación crea la primera conexión de cliente determina el valor que se utiliza cuando la aplicación crea cualquier conexión de cliente posterior. Por lo tanto, cuando la aplicación crea una conexión de cliente posterior, no se tiene en cuenta el valor del campo sslFipsRequired. Debe reiniciar la aplicación si desea utilizar un valor diferente para el campo sslFipsRequired.

Para conectar correctamente mediante TLS, el almacén de confianza de JSSE se debe configurar con certificados raíz de entidad emisora de certificados a partir de los cuales se puede autenticar el certificado presentado por el gestor de colas. Similarmente, si SSLClientAuth en el canal SVRCONN se ha establecido en MQSSL_CLIENT_AUTH_REQUIRED, el almacén de claves de JSSE debe contener un certificado de identificación que sea de confianza para el gestor de colas.

Referencia relacionada
Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Utilización del nombre distinguido del gestor de colas en IBM MQ classes for Java
El gestor de colas acredita su identidad utilizando un certificado TLS, el cual contiene un nombre distinguido (DN). Una aplicación cliente de IBM MQ classes for Java puede utilizar este nombre distinguido para asegurarse de que se está comunicando con el gestor de colas correcto.

Se especifica un patrón de DN utilizando la variable sslPeerName de MQEnvironment. Por ejemplo, si establece:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permite que la conexión se realice correctamente sólo si el gestor de colas presenta un certificado con un nombre común que empieza por QMGR., y al menos dos nombres de unidad organizativa, el primero de los cuales debe ser IBM y el segundo WebSphere.

Si se define sslPeerName, la conexión sólo se establece si el valor de ese parámetro es un patrón válido y el gestor de colas presenta el certificado correspondiente.

Una aplicación puede también especificar el nombre distinguido del gestor de colas estableciendo la propiedad de entorno CMQC.SSL_PEER_NAME_PROPERTY. Para obtener más información sobre los nombres distinguidos, consulte Nombres distinguidos.

Utilización de listas de revocación de certificados en IBM MQ classes for Java
Especifique las listas de revocación de certificados que se deben utilizar mediante java.security.cert.CertStore class. Entonces IBM MQ classes for Java comparará los certificados presentados con la lista de revocación de certificados especificada.

Una lista de revocación de certificados (CRL) es un conjunto de certificados que han sido revocados, ya sea por la entidad emisora de certificados o por la organización local. Las CRL normalmente se alojan en servidores LDAP. En Java 2 v1.4, se puede especificar un servidor de CRL en tiempo de conexión y el certificado presentado por el gestor de colas se compara con la CRL antes de permitir la conexión. Para obtener más información sobre las listas de revocación de certificados y IBM MQ, consulte Utilización de listas de revocación de certificados y listas de revocación de autorizaciones y Acceso a las CRL y a las ARL en IBM MQ classes for Java y IBM MQ classes for JMS.

Nota: Para utilizar un CertStore correctamente con una CRL alojada en un servidor LDAP, asegúrese de que el SDK (Software Development Kit) de Java es compatible con la CRL. Algunos SDK necesitan que la CRL cumpla el RFC 2587, el cual define un esquema para LDAP v2. La mayoría de servidores LDAP v3 utilizan el RFC 2256.

Las CRL que se deben utilizar se especifican mediante la clase java.security.cert.CertStore. Consulte la documentación sobre esta clase para obtener información detallada sobre cómo obtener instancias de CertStore. Para crear un CertStore basado en un servidor LDAP, primero cree una instancia de LDAPCertStoreParameters, inicializada con los valores de puerto y de servidor que se deben utilizar. Por ejemplo:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Después de crear una instancia de CertStoreParameters, utilice el constructor estático de CertStore para crear un CertStore de tipo LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

También se da soporte a otros tipos de CertStore (por ejemplo, recopilación). Habitualmente, hay varios servidores de CRL configurados con información de CRL idéntica para proporcionar redundancia. Cuando tenga un objeto CertStore para cada uno de estos servidores de CRL, coloque todos los objetos en una colección adecuada. El ejemplo siguiente muestra los objetos CertStore colocados en una ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Esta recopilación se puede establecer en la variable estática de MQEnvironment sslCertStores, antes de conectarse para habilitar la comprobación de CRL:

```
MQEnvironment.sslCertStores = crls;
```

El certificado que presenta el gestor de colas en el momento de establecer una conexión, se valida de la forma siguiente:

1. El primer objeto CertStore de la recopilación identificado como sslCertStores se utiliza para identificar un servidor CRL.
2. Se hace un intento de contactar con el servidor CRL.
3. Si resulta satisfactorio, se busca una coincidencia del certificado en el servidor.
   a. Si se encuentra el certificado para revocarlo, finaliza el proceso de búsqueda y la petición de conexión no responde indicando el código de razón MQRC_SSL_CERTIFICATE_REVOKED.
   b. Si no se encuentra el certificado, finaliza el proceso de búsqueda y se permite que la conexión continúe.
4. Si el intento de contactar con el servidor no resulta satisfactorio, se utiliza el siguiente objeto CertStore para identificar un servidor CRL y se repite el proceso desde el paso 2.

   Si este es el último CertStore de la colección o si la colección no contiene objetos CertStore, el proceso de búsqueda ha fallado y la solicitud de conexión falla y devuelve el código de razón MQRC_SSL_CERT_STORE_ERROR.

El objeto Collection determina el orden en el que se utilizan los CertStores.

La colección de CertStores también se puede establecer mediante CMQC.SSL_CERT_STORE_PROPERTY. Por comodidad, esta propiedad también permite especificar un solo CertStore sin que sea miembro de una colección.

Si sslCertStores se establece en un valor nulo, no se efectúa ninguna comprobación de CRL. Esta propiedad no se tiene en cuenta si sslCipherSuite no está establecido.

Renegociación de la clave secreta en IBM MQ classes for Java
Una aplicación cliente de IBM MQ classes for Java puede controlar cuándo se renegocia la clave secreta que se utiliza para el cifrado en una conexión de cliente, respecto al número total de bytes enviados y recibidos.

La aplicación puede hacerlo de una de las dos formas siguientes: si la aplicación utiliza más de uno de estos procedimientos, se aplican las reglas de prioridad habituales.

- Estableciendo el campo sslResetCount en la clase MQEnvironment.
- Estableciendo la propiedad de entorno MQC.SSL_RESET_COUNT_PROPERTY en un objeto Hashtable. A continuación, la aplicación asigna la tabla hash al campo `properties` en la clase MQEnvironment o pasa la tabla hash a un objeto MQQueueManager de su constructor.

El valor del campo sslResetCount o de la propiedad de entorno MQC.SSL_RESET_COUNT_PROPERTY representa el número total de bytes enviados y recibidos por el código de cliente IBM MQ classes for Java antes de que se renegocie la clave secreta. El número de bytes enviados es el número antes del cifrado y el número de bytes recibidos es el número después del cifrado. El número de bytes incluye también la información de control enviada y recibida por el cliente IBM MQ classes for Java.

Si la cuenta de restablecimiento es cero, que es el valor predeterminado, la clave secreta nunca se renegocia. La cuenta de restablecimiento se ignora si no se especifica ninguna CipherSuite.

Suministro de una SSLSocketFactory personalizada en IBM MQ classes for Java
Si utiliza una fábrica de sockets JSSE personalizada, establezca MQEnvironment.sslSocketFactory en el objeto de fábrica personalizado. Los detalles dependiendo de cada implementación de JSSE.

Diferentes implementaciones de JSSE pueden proporcionar características diferentes. Por ejemplo, una implementación de JSSE especializada podría permitir la configuración de un modelo determinado de hardware de cifrado. Además, algunos proveedores JSSE permiten que un programa personalice almacenes de claves y almacenes de confianza o permiten modificar la elección del certificado de identidad del almacén de claves. En JSSE, todas estas personalizaciones se integran en una clase de fábrica denominada javax.net.ssl.SSLSocketFactory.

Consulte la documentación de JSSE para obtener detalles sobre cómo crear una implementación personalizada de SSLSocketFactory. Los detalles varían de un proveedor a otro, pero los pasos habituales podrían ser estos:

1. Cree un objeto SSLContext mediante un método estático en SSLContext
2. Inicialice este SSLContext con implementaciones adecuadas de KeyManager y TrustManager (creadas a partir de sus propias clases de fábrica)
3. Cree una SSLSocketFactory a partir de SSLContext

Cuando tenga un objeto SSLSocketFactory, establezca el valor de MQEnvironment.sslSocketFactory en el objeto de fábrica personalizado. Por ejemplo:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java utiliza esta SSLSocketFactory para conectar con el gestor de colas de IBM MQ. Esta propiedad también se puede establecer utilizando CMQC.SSL_SOCKET_FACTORY_PROPERTY. Si sslSocketFactory se establece en un valor nulo, se utiliza el valor predeterminado de SSLSocketFactory de la JVM. Esta propiedad no se tiene en cuenta si sslCipherSuite no está establecido.

Cuando utilice SSLSocketFactories personalizadas, tenga en cuenta el efecto del uso compartido de conexiones TCP/IP. Si el uso compartido de conexiones está habilitado, no se solicita un nuevo socket de la SSLSocketFactory proporcionada, incluso cuando el socket producido sería distinto de alguna manera en el contexto de una solicitud de conexión posterior. Por ejemplo, si se debe presentar un certificado de cliente diferente en una conexión posterior, no se debe permitir el uso compartido de conexiones.

Realización de cambios en el almacén de claves o almacén de confianza de JSSE en IBM MQ classes for Java
Si cambia el almacén de claves o almacén de confianza de JSSE, debe realizar las acciones siguientes para que los cambios surtan efecto.

Si cambia el contenido del almacén de claves o del almacén de confianza de JSSE, o cambia la ubicación del archivo de almacén de claves o de almacén de confianza, las aplicaciones de IBM MQ classes for Java que se estén ejecutando en ese momento no recogerán automáticamente los cambios. Para que los cambios surtan efecto, deben realizarse las acciones siguientes:

- Las aplicaciones deben cerrar todas sus conexiones y eliminar cualquier conexión sin utilizar en las agrupaciones de conexiones.
- Si el proveedor JSSE almacena información del almacén de claves y almacén de confianza, esta información se debe actualizar.

Una vez realizadas estas acciones, las aplicaciones pueden volver a crear sus conexiones.

Dependiendo de cómo estén diseñadas las aplicaciones y de la función proporcionada por el proveedor JSSE, puede ser posible realizar estas acciones sin detener y reiniciar las aplicaciones. Pero detener y reiniciar las aplicaciones puede ser la solución más sencilla.

Manejo de errores al utilizar TLS con IBM MQ classes for Java
IBM MQ classes for Java puede emitir diversos códigos de razón cuando se conecta a un gestor de colas mediante TLS.

Esos códigos se describen en la lista siguiente:

MQRC_SSL_NOT_ALLOWED
Estaba establecida la propiedad sslCipherSuite, pero se ha utilizado una conexión de enlaces. Sólo la conexión de cliente permite utilizar TLS.

MQRC_JSSE_ERROR
El proveedor de JSSE ha notificado un error que IBM MQ no ha podido tratar. Este error puede ser debido a un problema de configuración con JSSE, o a que el certificado presentado por el gestor de colas no se ha podido validar. La excepción producida por JSSE se puede recuperar mediante el método getCause() en MQException.

MQRC_SSL_INITIALIZATION_ERROR
Se ha emitido una llamada MQCONN o MQCONNX con las opciones de configuración TLS especificadas, pero se ha producido un error durante la inicialización del entorno de TLS.

MQRC_SSL_PEER_NAME_MISMATCH
El patrón de nombre distinguido especificado en la propiedad sslPeerName no coincidía con el nombre distinguido presentado por el gestor de colas.

MQRC_SSL_PEER_NAME_ERROR
El patrón de nombre distinguido especificado en la propiedad sslPeerName no era válido.

MQRC_UNSUPPORTED_CIPHER_SUITE
El proveedor de JSSE no ha reconocido la CipherSuite especificada en sslCipherSuite. Un programa puede obtener una lista completa de CipherSuites soportadas por el proveedor de JSSE mediante el método SSLSocketFactory.getSupportedCipherSuites(). Puede encontrar una lista de CipherSuites que se pueden utilizar para comunicarse con IBM MQ en "CipherSpecs y CipherSuites de TLS en IBM MQ classes for Java" en la página 420.

MQRC_SSL_CERTIFICATE_REVOKED
El certificado presentado por el gestor de colas se ha encontrado en una lista de revocación de certificados especificada por la propiedad sslCertStores. Actualice el gestor de colas para utilizar certificados de confianza.

MQRC_SSL_CERT_STORE_ERROR
No se ha podido buscar el certificado presentado por el gestor de colas en ninguno de los CertStores proporcionados. El método MQException.getCause() devuelve el error que se ha producido al buscar en el primer CertStore probado. Si la excepción causal es NoSuchElementException, ClassCastException o NullPointerException, compruebe que la colección especificada en la propiedad sslCertStores contenga como mínimo un objeto CertStore válido.

CipherSpecs y CipherSuites de TLS en IBM MQ classes for Java
La capacidad de las aplicaciones de IBM MQ classes for Java para establecer conexiones con un gestor de colas depende de la suite de cifrado especificada en el extremo servidor del canal MQI y de la suite de cifrado especificada en el extremo cliente.

La tabla siguiente lista las especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes.

Deprecated Revise el tema CipherSpecs en desuso para ver si alguna de las especificaciones de cifrado listadas en la tabla siguiente ha dejado de ser utilizada por IBM MQ y, si es así, en qué actualización se ha dejado de utilizar la especificación de cifrado.

Importante: Las CipherSuites listadas son las soportadas por el IBM Java Runtime Environment (JRE) proporcionado con IBM MQ. Las suites de cifrado indicadas en la lista incluyen las soportadas por el Java JRE de Oracle. Para obtener más información sobre cómo configurar la aplicación para que utilice un Java JRE de Oracle, consulte "Configuración de la aplicación para utilizar correlaciones de las suites de cifrado de IBM Java u Oracle Java" en la página 442.

La tabla también incluye el protocolo utilizado por la aplicación y la indicación de si la suite de cifrado cumple el estándar FIPS 140-2.

Nota: En AIX, Linux, and Windows, IBM MQ proporciona conformidad con FIPS 140-2 a través del módulo criptográfico IBM Crypto for C (ICC) . El certificado para este módulo se ha movido al estado Histórico. Los clientes deben ver el certificado deIBM Crypto for C (ICC) y tener en cuenta cualquier consejo proporcionado por NIST. Un módulo FIPS 140-3 de sustitución está actualmente en curso y su estado se puede ver buscándolo en los módulos NIST CMVP en la lista de procesos.

Las suites de cifrado designadas como compatibles con FIPS 140-2 se pueden utilizar si la aplicación no se ha configurado para aplicar la conformidad con FIPS 140-2, pero si se ha configurado la conformidad con FIPS 140-2 para la aplicación (consulte las notas siguientes sobre la configuración) sólo se pueden configurar las suites de cifrado marcadas como compatibles con FIPS 140-2; el intento de utilizar otras suites de cifrado produce un error.

Nota: Cada JRE puede tener varios proveedores de seguridad criptográfica, cada uno de los cuales puede contribuir a una implementación de la misma CipherSuite. Pero no todos los proveedores de seguridad están certificados como compatibles con FIPS 140-2. Si la conformidad con FIPS 140-2 no se aplica para una aplicación, es posible que se utilice una implementación no certificada de la suite de cifrado. Las implementaciones no certificadas pueden no trabajar en conformidad con FIPS 140-2, incluso si la suite de cifrado cumple teóricamente el nivel de seguridad mínimo exigido por el estándar. Consulte las notas siguientes para obtener más información sobre cómo configurar la imposición de FIPS 140-2 en las aplicaciones IBM MQ Java.

Para obtener más información acerca de la conformidad con FIPS 140-2 y Suite-B para CipherSpecs y CipherSuites, consulte Especificación de CipherSpecs. También puede ser necesario que conozca información que se refiere a los Estándares federales de proceso de información de los Estados Unidos.

Para utilizar el conjunto completo de suites de cifrado y trabajar en conformidad con FIPS 140-2 o Suite-B, es necesario un JRE adecuado. adecuado.IBM Java 7 Service Refresh 4 Fixpack 2 o un nivel superior de IBM JRE proporciona el soporte adecuado para las CipherSuites TLS 1.2 listadas en Tabla 59 en la página 422.

Para poder utilizar cifrados TLS 1.3 , el JRE que ejecuta la aplicación debe dar soporte a TLS 1.3.

Nota: Para utilizar algunas suites de cifrado, es necesario configurar los archivos de política 'no restringidos' en el JRE. Para obtener más detalles sobre cómo se configuran los archivos de políticas en un SDK o JRE, consulte el tema *Archivos de políticas SDK de IBM* en la publicación *Security Reference for IBM SDK, Java Technology Edition* correspondiente a la versión que está utilizando.

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	No
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	sí

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	sí
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA "2" en la página 441	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	no "4" en la página 441

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	no "4" en la página 441
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	no "4" en la página 441

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	no "4" en la página 441
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	no "4" en la página 441

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	no "4" en la página 441
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	no "4" en la página 441

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	No
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	No
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	sí
TLS_AES_128_GCM_SHA256 "3" en la página 441	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_AES_256_GCM_SHA384 "3" en la página 441	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	No
TLS_CHACHA20_POLY1305_SHA256 "3" en la página 441	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
TLS_AES_128_CCM_SHA256 "3" en la página 441	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	No
TLS_AES_128_CCM_8_SHA256 "3" en la página 441	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	No
Cualquiera "3" en la página 441	*ANY	*ANY	Múltiple	No
ANY_TLS13 "3" en la página 441	*TLS13	*TLS13	TLS V13	No

Tabla 59. Especificaciones de cifrado soportadas por IBM MQ y sus suites de cifrado equivalentes (continuación)

CipherSpec **"1" en la página 441**	**Suite de cifrado equivalente (IBM JRE)**	**Suite de cifrado equivalente (Oracle JRE)**	**Protocolo**	**Compatible con FIPS 140-2**
ANY_TLS12_OR_HIGHER "3" en la página 441	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 y superior	No
ANY_TLS13_OR_HIGHER "3" en la página 441	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 y superior	No

Notas:

1. Este es el valor configurado en un canal en IBM MQ, incluido en una CCDT (binario o JSON).
2. Deprecated La CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.
3. Para poder utilizar cifrados TLS v1.3 , el Java runtime environment (JRE) que ejecuta la aplicación debe dar soporte a TLS v1.3.
4. V 9.3.0.17 V 9.3.5.1 A partir de IBM MQ 9.3.5 CSU 1 y IBM MQ 9.3.0 CSU 17, el JRE de IBM Java 8 elimina el soporte para el intercambio de claves RSA cuando opera en modalidad FIPS.

Configuración de suites de cifrado y la conformidad con FIPS en una aplicación de IBM MQ classes for Java

- Una aplicación que utiliza IBM MQ classes for Java puede utilizar cualquiera de dos métodos para establecer la suite de cifrado para una conexión:
  - Puede definir el campo sslCipherSuite con el nombre de la suite de cifrado en la clase MQEnvironment.
  - Puede definir la propiedad CMQC.SSL_CIPHER_SUITE_PROPERTY con el nombre de la suite de cifrado en la tabla hash de propiedades que se pasa al constructor MQQueueManager.
- Una aplicación que utiliza IBM MQ classes for Java puede utilizar cualquiera de dos métodos para aplicar la conformidad con FIPS 140-2:
  - Puede establecer el campo sslFipsRequired en true en la clase MQEnvironment.
  - Puede establecer la propiedad CMQC.SSL_FIPS_REQUIRED_PROPERTY en true en la tabla hash de propiedades que se pasa al constructor MQQueueManager.

Configuración de la aplicación para utilizar correlaciones de las suites de cifrado de IBM Java u Oracle Java

Nota:

V 9.3.3 Para Continuous Delivery de IBM MQ 9.3.3, la Java Propiedad del sistema `com.ibm.mq.cfg.useIBMCipherMappings`, que controla qué correlaciones se utilizan, se elimina del producto. A partir de IBM MQ 9.3.3, un cifrado se puede definir como el nombre CipherSpec o CipherSuite y IBM MQlo maneja correctamente. Los tres archivos JAR Jackson siguientes son necesarios si la aplicación IBM MQ classes for Java crea conexiones TLS seguras con un gestor de colas:

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

Importante: La información siguiente sobre `com.ibm.mq.cfg.useIBMCipherMappings` se aplica únicamente a Long Term Support y Continuous Delivery antes de IBM MQ 9.3.3 .

Puede configurar si la aplicación utiliza las correlaciones predeterminadas de suites de cifrado de IBM Java con especificaciones de cifrado de IBM MQ, o las correlaciones de suites de cifrado de Oracle con especificaciones de cifrado de IBM MQ. Por lo tanto, puede utilizar suites de cifrado de TLS si la aplicación utiliza un JRE de IBM o un JRE de Oracle. La Java Propiedad del sistema `com.ibm.mq.cfg.useIBMCipherMappings` controla qué correlaciones se utilizan. La propiedad puede tener uno de los valores siguientes:

true
Utilizar las correlaciones de suites de cifrado de IBM Java con especificaciones de cifrado de IBM MQ.

Este es el valor predeterminado.

falso
Utilizar las correlaciones de suites de cifrado de Oracle con especificaciones de cifrado de IBM MQ.

Para obtener más información sobre cómo utilizar IBM MQ Java y los cifrados TLS, consulte la publicación de blog de MQdev MQ Java, Cifrados TLS, NoIBM JRE & APARs IT06775, IV66840, IT09423, IT10837.

Limitaciones de interoperatividad

Determinadas suites de cifrado pueden ser compatibles con más de una especificación de cifrado de IBM MQ, dependiendo del protocolo que se esté utilizando. Pero solo está soportada la combinación CipherSuite/CipherSpec que utiliza la versión de TLS especificada en la Tabla 1. El intento de utilizar combinaciones no soportadas de suites de cifrado y especificaciones de cifrado producirá un error y se devolverá la excepción correspondiente. Las instalaciones que utilicen cualquiera de estas combinaciones de suite de cifrado/especificación de cifrado se deben trasladar a una combinación soportada.

La tabla siguiente muestra las suites de cifrado a la que se aplica esta limitación.

Tabla 60. Suites de cifrado y sus correspondientes especificaciones de cifrado soportadas y no soportadas

CipherSuite	Especificación de cifrado soportada para TLS	Especificación de cifrado no soportada para TLS
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" en la página 443	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Nota:

1. Deprecated La especificación de cifrado TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.

Ejecución de aplicaciones de IBM MQ classes for Java

Si escribe una aplicación (una clase que contiene un método main()) utilizando la modalidad de enlaces o de cliente, ejecute el programa utilizando el intérprete de Java.

Utilice el mandato:

```
java -Djava.library.path= library_path MyClass
```

donde *vía_acceso_bibliotecas* es la vía de acceso de las bibliotecas de IBM MQ classes for Java. Para obtener más información, consulte "IBM MQ classes for Java bibliotecas" en la página 367.

Tareas relacionadas

Rastreo de aplicaciones de IBM MQ classes for Java

Rastreo del adaptador de recursos de IBM MQ

Comportamiento dependiente del entorno de IBM MQ classes for Java

IBM MQ classes for Java le permite crear aplicaciones que se pueden ejecutar en distintas versiones de IBM MQ. Esta colección de temas describe el comportamiento de las clases de Java que dependen de estas distintas versiones.

IBM MQ classes for Java proporciona un conjunto de clases principales, que proporcionan una función y un comportamiento coherentes en todos los entornos. Las funciones disponibles fuera de estas clases principales dependen de la funcionalidad del gestor de colas al que está conectado la aplicación.

Salvo que se indique otra cosa, el comportamiento mostrado es el que se describe en la sección Referencia de aplicaciones MQI correspondiente al gestor de colas.

Clases principales en IBM MQ classes for Java

IBM MQ classes for Java contiene un conjunto principal de clases, que se pueden utilizar en todos los entornos.

Las clases siguientes se consideran clases principales y se pueden utilizar en todos los entornos, con solamente las variaciones secundarias que se listan en "Restricciones y variaciones para clases principales de IBM MQ classes for Java" en la página 445.

- MQEnvironment
- MQException

- MQGetMessageOptions

  Excepto:

  – MatchOptions
  – GroupStatus
  – SegmentStatus
  – Segmentación
- MQManagedObject

  Excepto:

  – inquire()
  – set()
- MQMessage

  Excepto:

  – groupId
  – messageFlags
  – messageSequenceNumber
  – offset
  – originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  Excepto:

  – knownDestCount
  – unknownDestCount
  – invalidDestCount
  – recordFields
- MQProcess
- MQQueue
- MQQueueManager

  Excepto:

  – begin()
  – accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

Nota:

1. Algunas constantes no se incluyen en el conjunto de clases principales (consulte “Restricciones y variaciones para clases principales de IBM MQ classes for Java” en la página 445 para conocer detalles). No utilice estas constantes en programas completamente portátiles.
2. Algunas plataformas no son compatibles con todas las modalidades de conexión. En estas plataformas, sólo puede utilizar las clases principales y las opciones relacionadas con las modalidades soportadas.

Restricciones y variaciones para clases principales de IBM MQ classes for Java

Las clases principales generalmente se comportan de forma homogénea en todos los entornos, incluso si las llamadas MQI equivalentes tienen normalmente diferencias de entorno. El comportamiento es como si se utilizara un gestor de colas AIX, Linuxo Windows , excepto para las siguientes restricciones y variaciones menores.

Restricciones para los valores MQGMO_* en IBM MQ classes for Java
Determinados valores MQGMO_* no son compatibles con todos los gestores de colas.

El uso de los valores MQGMO_* siguientes puede hacer MQQueue.get() emita una excepción:

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

Además, MQGMO_SET_SIGNAL no se puede utilizar desde Java.

Restricciones para los valores MQPMRF_* en IBM MQ classes for Java
Estos valores sólo se utilizan cuando se transfieren mensajes a una lista de distribución, y sólo están soportados por los gestores de colas que son compatibles con listas de distribución. Por ejemplo, los gestores de colas de z/OS no son compatibles con listas de distribución.

Restricciones para los valores MQPMO_* en IBM MQ classes for Java
Determinados valores MQPMO_* no son compatibles con todos los gestores de colas

El uso de los valores MQPMO_* siguientes puede hacer que MQQueue.put() o MQQueueManager.put() emita una excepción:

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

Restricciones y variaciones para los valores MQCNO_* en IBM MQ classes for Java
Determinados valores `MQCNO_*` no están soportados.

- La reconexión automática de cliente no está soportada por IBM MQ classes for Java. Cualquiera que sea el valor que establezca para `MQCNO_RECONNECT_*`, la conexión continúa comportándose como si se hubiera establecido `MQCNO_RECONNECT_DISABLED`.
- `MQCNO_FASTPATH` no tiene ningún efecto en los gestores de colas que no son compatibles con `MQCNO_FASTPATH`. Tampoco tiene ningún efecto en las conexiones de cliente.

Restricciones para los valores MQRO_* en IBM MQ classes for Java
Se pueden establecer las opciones de informe siguientes:

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

Para obtener más información, consulte Informe.

Diferencias diversas entre IBM MQ classes for Java en z/OS y otras plataformas
IBM MQ for z/OS se comporta de forma distinta que IBM MQ en otras plataformas en algunos aspectos.

BackoutCount
Un gestor de colas de z/OS devuelve un número máximo de restituciones de 255, incluso si el mensaje se ha restituido más de 255 veces.

Prefijo de cola dinámica predeterminado
Cuando se conecta a un gestor de colas de z/OS utilizando una conexión de enlaces, el prefijo de cola dinámica predeterminado es CSQ.*. En otro caso, el prefijo de cola dinámica predeterminado es AMQ.*.

Constructor MQQueueManager
La conexión de cliente no está soportada en z/OS. El intentar conectar con opciones de cliente produce una MQException con MQCC_FAILED y MQRC_ENVIRONMENT_ERROR.

El constructor MQQueueManager puede también fallar y devolver MQRC_CHAR_CONVERSION_ERROR (si no puede inicializar la conversión entre las páginas de códigos IBM-1047 e ISO8859-1), o MQRC_UCS2_CONVERSION_ERROR (si no puede inicializar la conversión entre la página de códigos del gestor de colas y Unicode). Si la aplicación falla con uno de estos códigos de razón, asegúrese de que se ha instalado el componente National Language Resources de Language Environment y asegúrese de que están disponibles las tablas de conversión correctas.

Las tablas de conversión para Unicode se instalan como parte del componente opcional de C/C++ para z/OS. Consulte la publicación *z/OS C/C++ Programming Guide*, SC09-4765, para obtener más información sobre cómo habilitar las conversiones UCS-2.

Funciones disponibles fuera de las clases principales de IBM MQ classes for Java

IBM MQ classes for Java contiene determinadas funciones que están específicamente diseñadas para utilizar extensiones de API que no están soportadas por todos los gestores de colas. Esta colección de temas describe cómo se comportan estas funciones cuando se utiliza un gestor de colas en el que no están soportadas.

Variaciones en la opción de constructor MQQueueManager
Algunos de los constructores MQQueueManager incluyen un argumento entero opcional. Algunos valores de este argumento no están permitidos en todas las plataformas.

Cuando un constructor MQQueueManager incluye un argumento entero opcional, se correlaciona con el campo de opciones MQCNO de la MQI y se utiliza para conmutar entre la conexión normal y la conexión de vía rápida. Esta forma ampliada del constructor se acepta en todos los entornos si las únicas opciones utilizadas son MQCNO_STANDARD_BINDING o MQCNO_FASTPATH_BINDING. Cualquier otra opción hace que el constructor falle y devuelva un error MQRC_OPTIONS_ERROR. La opción de conexión de vía rápida CMQC.MQCNO_FASTPATH_BINDING sólo se tiene en cuenta para una conexión de enlaces con un gestor de colas que soporte esa conexión. En otros entornos, esa opción no se tiene en cuenta.

Restricciones en el método MQQueueManager.begin()
Este método sólo se puede utilizar en un gestor de colas IBM MQ en sistemas AIX, Linux, and Windows en modalidad de enlaces. En otro caso, el método falla y devuelve el error MQRC_ENVIRONMENT_ERROR.

Consulte “Coordinación de JTA/JDBC utilizando IBM MQ classes for Java” en la página 411 para obtener más detalles.

Variaciones en los campos MQGetMessageOptions
Algunos gestores de colas no son compatibles con la versión 2 de la estructura de MQGMO, por lo que algunos campos se deben establecer en los valores predeterminados.

Cuando utilice un gestor de colas que no sea compatible con versión 2 de la estructura MQGMO, deje los campos siguientes establecidos en sus valores predeterminados:

GroupStatus
SegmentStatus
Segmentación

Además, el campo MatchOptions solo permite utilizar MQMO_MATCH_MSG_ID y MQMO_MATCH_CORREL_ID. Si transfiere valores no permitidos a estos campos, la operación MQDestination.get() subsiguiente falla y devuelve MQRC_GMO_ERROR. Si el gestor de colas no es compatible con la versión 2 de la estructura MQGMO, estos campos no se actualizan después de una operación MQDestination.get() satisfactoria.

Restricciones en las listas de distribución en IBM MQ classes for Java
No todos los gestores de colas permiten abrir MQDistributionList.

Las clases siguientes se utilizan para crear listas de distribución:

MQDistributionList
MQDistributionListItem
MQMessageTracker

Puede crear y llenar con datos MQDistributionList y MQDistributionListItem en cualquier entorno, pero no todos los gestores de colas permiten abrir una MQDistributionList. En particular, los gestores de colas de z/OS no permiten el uso de listas de distribución. Si intenta abrir una MQDistributionList cuando se utiliza este tipo de gestor de colas, se genera un MQRC_OD_ERROR.

Variaciones en los campos de MQPutMessageOptions
Si un gestor de colas no es compatible con las listas de distribución, determinados campos de MQPMO se tratan de forma diferente.

Cuatro campos de MQPMO se representan como las variables de miembro siguientes en la clase MQPutMessageOptions:

knownDestCount
unknownDestCount
invalidDestCount
recordFields

Estos campos se han diseñado principalmente para ser utilizados con listas de distribución. Pero un gestor de colas que sea compatible con listas de distribución también llena los campos DestCount después de una operación MQPUT sobre una cola individual. Por ejemplo, si la resolución de la cola da como resultado una cola local, knownDestCount se establece en 1 y los otros dos campos de recuento se establecen en 0.

Si el gestor de colas no es compatible con listas de distribución, estos valores se simulan de la forma siguiente:

- Si put() se ejecuta correctamente, unknownDestCount se establece en 1 y los demás campos se establecen en 0.
- Si put() falla, invalidDestCount se establece en 1 y los demás campos se establecen en 0.

La variable recordFields se utiliza con listas de distribución. Se puede escribir un valor en recordFields en cualquier momento, sin importar el entorno utilizado. Ese campo no se tiene en cuenta si se utiliza el objeto MQPutMessageOptions en una operación subsiguiente MQDestination.put() o MQQueueManager.put(), en lugar de utilizar MQDistributionList.put().

Restricciones en los campos de MQMD en IBM MQ classes for Java
Algunos campos de MQMD relacionados con las segmentación de mensajes se deben dejar en su valor predeterminado cuando se utiliza un gestor de colas que no es compatible con la segmentación.

Los campos de MQMD siguientes afectan principalmente a la segmentación de mensajes:

GroupId
MsgSeqNumber
Desplazamiento
MsgFlags
OriginalLength

Si una aplicación establece cualquiera de estos campos de MQMD en valores distintos de los valores predeterminados y luego efectúa una operación put() o get() en un gestor de colas que no es compatible con esos campos, put() o get() emite una excepción MQException y devuelven un error MQRC_MD_ERROR. Una operación put() o un get() ejecutada satisfactoriamente con un gestor de colas de este tipo siempre deja los campos de MQMD establecidos en sus valores predeterminados. No envíe un mensaje agrupado o segmentado a una aplicación Java que se ejecuta en un gestor de colas que no da soporte a la segmentación y agrupación de mensajes.

Si una aplicación Java intenta obtener un mensaje de un gestor de colas que no es compatible con estos campos, y el mensaje físico que se debe obtener forma parte de un grupo de mensajes segmentados (es decir, tiene valores distintos de los predeterminados para los campos de MQMD), el mensaje se obtiene sin error. Pero los campos de MQMD contenidos en MQMessage no se actualizan, la propiedad de formato de MQMessage se establece en MQFMT_MD_EXTENSION y los verdaderos datos del mensaje toman como prefijo una estructura MQMDE que contiene los valores para los nuevos campos.

Restricciones para IBM MQ classes for Java bajo CICS Transaction Server

En el entorno de CICS Transaction Server para z/OS, solamente la primera hebra (principal) puede emitir llamadas de CICS o IBM MQ.

Observe que las clases IBM MQ JMS no están soportadas para ser utilizadas en una aplicación CICS Java.

Por este motivo, no se pueden compartir objetos MQQueueManager o MQQueue entre hebras en este entorno, ni se puede crear un nuevo MQQueueManager en una hebra dependiente.

z/OS "Diferencias diversas entre IBM MQ classes for Java en z/OS y otras plataformas" en la página 446 identifica algunas restricciones y variaciones que se aplican a las IBM MQ classes for Java cuando se ejecutan para un gestor de colas de z/OS. Además, cuando las clases se ejecutan en CICS, los métodos de control de transacciones de MQQueueManager no están soportados. En lugar de emitir MQQueueManager.commit () o MQQueueManager.backout (), las aplicaciones utilizan los métodos de sincronización de tareas JCICS , Task.commit() y Task.rollback(). La clase Task la proporciona JCICS en el paquete com.ibm.cics.server .

Utilización del adaptador de recursos de IBM MQ

El adaptador de recursos permite que las aplicaciones que se ejecutan en un servidor de aplicaciones accedan a recursos de IBM MQ. El adaptador de recursos da soporte a la comunicación de entrada y de salida.

Contenido del adaptador de recursos

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. Además del adaptador de recursos que da soporte a Java EE y JMS 2.0, IBM MQ 9.3.0 proporciona un adaptador de recursos que da soporte a Jakarta Messaging.

JM 3.0 **Adaptador de recursos de IBM MQ para Jakarta Messaging**

El Jakarta Connectors Architecture proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno Jakarta EE a un EIS (Enterprise Information System) como, por ejemplo, IBM MQ o Db2. El adaptador de recursos de IBM MQ para Jakarta Messaging implementa las interfaces Jakarta Connectors 2.0.0 y contiene el IBM MQ classes for Jakarta Messaging. Permite que las aplicaciones Jakarta Messaging y los beans controlados por mensajes (MDB) que se ejecutan en un servidor de aplicaciones accedan a los recursos de un gestor de colas de IBM MQ . El adaptador de recursos soporta tanto el dominio punto a punto como el dominio de publicación/suscripción.

JMS 2.0 **Adaptador de recursos de IBM MQ para JMS 2.0**

La arquitectura Java Platform, Enterprise Edition Connector Architecture (JCA) proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno Java EE con un Enterprise Information System (EIS) tal como IBM MQ o Db2. El adaptador de recursos de IBM MQ para JMS 2.0 implementa las interfaces JCA 1.7 y contiene el IBM MQ classes for JMS. Permite que las aplicaciones de JMS y los beans controlados por mensajes (MDB) que se ejecutan en un servidor de aplicaciones accedan a los recursos de un gestor de colas de IBM MQ. El adaptador de recursos soporta tanto el dominio punto a punto como el dominio de publicación/suscripción.

El adaptador de recursos de IBM MQ da soporte a dos tipos de comunicación entre una aplicación y un gestor de colas:

Comunicación de salida

Una aplicación inicia una conexión con un gestor de colas y luego envía mensajes de JMS a destinos de JMS y recibe mensajes de JMS de destinos de JMS de forma síncrona.

Comunicación de entrada

Un mensaje de JMS que llega a un destino de JMS se entrega a un bean controlado por mensaje, que procesa el mensaje asíncronamente.

El adaptador de recursos también contiene las clases IBM MQ classes for Java. Estas clases pueden ser utilizadas automáticamente por aplicaciones que se ejecutan en un servidor de aplicaciones en el que se ha desplegado el adaptador de recursos, y permiten que las aplicaciones que se ejecutan en ese servidor de aplicaciones utilicen la API de IBM MQ classes for Java cuando acceden a recursos de un gestor de colas de IBM MQ.

El uso de las clases IBM MQ classes for Java dentro de un entorno Java EE está soportado con restricciones. Para obtener información sobre estas restricciones, consulte “Ejecución de aplicaciones de IBM MQ classes for Java en Java EE” en la página 358.

Qué versión del adaptador de recursos se debe utilizar

V 9.3.0 V 9.3.0 La versión del adaptador de recursos que utilice dependerá de si lo está desplegando en un servidor de aplicaciones que dé soporte a Jakarta EE o Java EE:

V 9.3.0 JM 3.0 V 9.3.0 **Jakarta EE**

A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 . El adaptador de recursos de IBM MQ para Jakarta Messaging debe desplegarse en un servidor de aplicaciones que dé soporte a Jakarta EE.

Java EE 7

El IBM MQ 8.0 y el adaptador de recursos posterior admite JCA v1.7 y proporciona soporte de JMS 2.0. Este adaptador de recursos se debe desplegar dentro de un servidor de aplicaciones de Java EE 7 o versión posterior (consulte “Sentencia de soporte del adaptador de recursos de IBM MQ” en la página 450).

Puede instalar el adaptador de recursos de IBM MQ 8.0 o versión posterior en cualquier servidor de aplicaciones que esté certificado como compatible con la especificación de Java Platform, Enterprise Edition 7. Utilizando el adaptador de recursos IBM MQ 8.0 o posterior, una aplicación puede conectarse a un gestor de colas utilizando el transporte BINDINGS o CLIENT.

Importante: El adaptador de recursos de IBM MQ 8.0 o posterior sólo se puede desplegar en un servidor de aplicaciones que dé soporte a JMS 2.0.

Utilización del adaptador de recursos con WebSphere Application Server traditional

A partir de IBM MQ 9.0, el adaptador de recursos IBM MQ está preinstalado en WebSphere Application Server traditional 9.0 o posterior. Por lo tanto, no es necesario instalar un adaptador de recursos nuevo.

JM 3.0 WebSphere Application Server traditional no da soporte actualmente a Jakarta EE. Consulte Declaración de soporte del adaptador de recursos deIBM MQ.

Nota: Un IBM MQ 9.0 o un adaptador de recursos posterior se puede conectar en la modalidad de transporte CLIENT o BINDINGS con cualquier gestor de colas IBM MQ en servicio.

Utilización del adaptador de recursos con WebSphere Liberty

Para conectar con IBM MQ desde WebSphere Liberty, debe utilizar el adaptador de recursos de IBM MQ. Puesto que Liberty no contiene el adaptador de recursos IBM MQ, debe obtenerlo por separado del Fix Central.

V 9.3.0 V 9.3.0 La versión del adaptador de recursos que utilice dependerá de si lo está desplegando en una versión de Liberty que admita Jakarta EE o Java EE.

Para obtener más información sobre cómo descargar e instalar el adaptador de recursos, consulte “Instalación del adaptador de recursos en Liberty” en la página 458.

Conceptos relacionados

“Configuración del adaptador de recursos para la comunicación de entrada” en la página 465
Para configurar la comunicación de entrada, defina las propiedades de uno o más objetos ActivationSpec.

“Configuración del adaptador de recursos para la comunicación de salida” en la página 485
Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

“Utilización de IBM MQ classes for JMS/Jakarta Messaging” en la página 85
IBM MQ classes for JMS y IBM MQ classes for Jakarta Messaging son los proveedores de mensajería de Java que se proporcionan con IBM MQ. Además de implementar las interfaces definidas en las especificaciones JMS y Jakarta Messaging , estos proveedores de mensajería añaden dos conjuntos de extensiones a la API de mensajería de Java .

“Utilización de IBM MQ classes for Java” en la página 356
Utilice IBM MQ en un entorno Java. IBM MQ classes for Java permite que una aplicación Java se conecte con IBM MQ como un cliente IBM MQ o se conecte directamente a un gestor de colas de IBM MQ.

Referencia relacionada

Configurar el servidor de aplicaciones para que utilice el último nivel de mantenimiento del adaptador de recursos

Determinación de problemas para el adaptador de recursos IBM MQ

Temas de WebSphere Application Server

Mantenimiento del adaptador de recursos de IBM MQ

Despliegue de aplicaciones JMS en Liberty para utilizar el proveedor de mensajería IBM MQ

Sentencia de soporte del adaptador de recursos de IBM MQ

El adaptador de recursos IBM MQ que debe utilizar para la comunicación entre una aplicación y un gestor de colas depende de si está utilizando la API Jakarta Messaging 3.0 o la API JMS 2.0 .

JMS 2.0 IBM MQ 8.0 o posterior se suministra con un adaptador de recursos que implementa la especificación JMS 2.0 . Solo se puede desplegar en un servidor de aplicaciones compatible con Java Platform, Enterprise Edition 7 (Java EE 7) y, por lo tanto, da soporte a JMS 2.0. Se mantiene una lista de servidores de aplicaciones certificados para Java Platform, Enterprise Edition en el sitio web deOracle.

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. Además del adaptador de recursos que da soporte a Java EE y JMS 2.0, IBM MQ 9.3.0 proporciona un adaptador de recursos que da soporte a Jakarta Messaging. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Puede obtener información adicional consultando Uso de IBM MQ classes for JMS.

Despliegue en WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 y posterior y WebSphere Application Server Liberty 9.0 y posterior son servidores de aplicaciones Java EE 7 certificados, de modo que el adaptador de recursos de IBM MQ 9.0 se puede desplegar en los mismos.

V 9.3.0 V 9.3.0 Para utilizar el adaptador de recursos de IBM MQ para Jakarta Messaging con Liberty, debe utilizar una versión de Liberty que dé soporte a Jakarta EE.

WebSphere Liberty tiene las siguientes características disponibles para trabajar con adaptadores de recursos:

- V 9.3.0 JM 3.0 V 9.3.0 La característica messaging-3.0 permite trabajar con adaptadores de recursos de Jakarta Messaging 3.0 .
- La característica wmqJmsClient-2.0 para permitir trabajar con adaptadores de recursos JMS 2.0.
- La característica wmqJmsClient-1.1 para permitir trabajar con adaptadores de recursos JMS 1.1.

Importante:

- V 9.3.0 JM 3.0 V 9.3.0 El adaptador de recursos de IBM MQ para Jakarta Messaging debe desplegarse en una versión de Liberty que admita Jakarta EE. Este adaptador de recursos no se puede utilizar con versiones de Liberty que den soporte a la especificación Java EE anterior, no Jakarta EE.
- JMS 2.0 El adaptador de recursos de IBM MQ 8.0 o posterior que da soporte a JMS 2.0 debe desplegarse con la característica wmqJmsClient-2.0 .

Despliegue en WebSphere Application Server traditional

WebSphere Application Server traditional 9.0 se proporciona con un adaptador de recursos IBM MQ 9.0 instalado. Por lo tanto, no es necesario instalar un adaptador de recursos nuevo. El adaptador de recursos instalado puede conectarse en modalidad de transporte CLIENT o BINDINGS a cualquier gestor de colas que se ejecute en una versión soportada de IBM MQ. Para obtener más información, consulte “Conectividad con gestores de colas de IBM MQ 8.0 o posteriores” en la página 452.

Importante:

- El adaptador de recursos de IBM MQ 9.0 no se puede desplegar en versiones de WebSphere Application Server traditional anteriores a IBM MQ 9.0, porque estas versiones no están certificadas por Java EE 7 .
- JM 3.0 WebSphere Application Server traditional no da soporte actualmente a Jakarta EE.

Para obtener más información sobre las versiones del adaptador de recursos que se suministran con WebSphere Application Server, consulte la nota técnica ¿Qué versión de WebSphere MQ Resource Adapter (RA) se suministra con WebSphere Application Server?

Utilización del adaptador de recursos con otros servidores de aplicaciones

Para todos los demás servidores de aplicaciones compatibles con Java EE 7 o Jakarta EE , los problemas que se producen después de la finalización satisfactoria de la prueba de verificación de la instalación (IVT) del adaptador de recursos de IBM MQ se pueden notificar a IBM para la investigación del rastreo del producto IBM MQ y otra información de diagnóstico de IBM MQ . Si la IVT del adaptador de recursos de IBM MQ no se puede ejecutar correctamente, es probable que los problemas encontrados se deban a un despliegue incorrecto o a definiciones de recursos incorrectas que sean específicas del servidor de aplicaciones y los problemas se deben investigar utilizando la documentación del servidor de aplicaciones y la organización de soporte para dicho servidor de aplicaciones.

Tiempo de ejecución de Java

El Java Runtime (JRE) que se utiliza para ejecutar el servidor de aplicaciones debe ser uno que esté soportado con el cliente IBM MQ 9.0 o posterior. Para obtener más información, consulte Requisitos del sistema para IBM MQ. (Seleccione el informe de versión y de sistema operativo o componente que desee ver y, a continuación, siga el enlace **Java** que aparece en la pestaña **Software soportado**).

Conectividad con gestores de colas de IBM MQ 8.0 o posteriores

El rango completo de funciones de JMS 2.0 está disponible cuando se conecta a un gestor de colas de IBM MQ 8.0 o posterior utilizando el adaptador de recursos que se ha desplegado en un servidor de aplicaciones certificado de Java EE 7 . Para utilizar la funcionalidad de JMS 2.0, el adaptador de recursos debe conectarse al gestor de colas utilizando la modalidad normal del proveedor de mensajería de IBM MQ. Para obtener más información, consulte Configuración de la propiedad de JMS **PROVIDERVERSION**.

El rango completo de funciones de Jakarta Messaging 3.0 está disponible cuando se conecta a un gestor de colas de IBM MQ 9.3 utilizando el adaptador de recursos que se ha desplegado en un servidor de aplicaciones certificado de Jakarta EE .

Extensiones MQ

La especificación JMS 2.0 incluye cambios en el funcionamiento de determinados comportamientos. Puesto que IBM MQ 8.0 o posterior implementa esta especificación, hay cambios en el comportamiento entre IBM MQ 8.0 y versiones posteriores y anteriores del producto. En IBM MQ 8.0 o posterior, IBM MQ classes for JMS incluyen el soporte para la propiedad del sistema Java `com.ibm.mq.jms.SupportMQExtensions` que, cuando se establece en TRUE, provoca que estas versiones de IBM MQ reviertan estos comportamientos a los de IBM WebSphere MQ 7.5 o anteriores. El valor predeterminado de la propiedad es FALSE.

El adaptador de recursos IBM MQ 9.0 o posterior también incluye una propiedad de adaptador de recursos llamada `supportMQExtensions` que tiene el mismo efecto y valor predeterminado que la propiedad del sistema `com.ibm.mq.jms.SupportMQExtensions` Java. De forma predeterminada, esta propiedad del adaptador de recursos está establecida en false en `ra.xml`.

Si tanto la propiedad del adaptador de recursos como la propiedad del sistema Java están establecidas, la propiedad del sistema tiene prioridad.

Tenga en cuenta que en el gestor de recursos que ya se ha desplegado en WebSphere Application Server traditional 9.0, esta propiedad se establece automáticamente en TRUE para ayudar al proceso de migración.

Para obtener más información, consulte “Propiedad SupportMQExtensions” en la página 337.

Problemas generales

No está soportada la intercalación de sesiones

Algunos servidores de aplicaciones proporcionan una función que se denomina intercalación de sesiones, en la que se puede utilizar la misma sesión JMS en varias transacciones, aunque solo se incluye en una cada vez. El adaptador de recursos de IBM MQ no da soporte a esta función, que puede generar los problemas siguientes:

```
Ha fallado un intento de colocar un mensaje en una cola MQ con el código de razón 2072
(MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Las llamadas a xa_close() falla con el código de razón -3 (XAER_PROTO), y se genera un FDC con el ID de sonda AT040010 en el gestor de colas IBM MQ al que se está accediendo desde el servidor de aplicaciones. Para obtener información sobre inhabilitar esta función, consulte la documentación del servidor de aplicaciones.

La especificación JTA (Java Transaction API) sobre cómo se recuperan los recursos XA para la recuperación de transacciones XA.

La sección 3.4.8 de la especificación JTA no define un mecanismo específico mediante el cual se vuelven a crear los recursos XA para realizar la recuperación de transacciones XA. Por lo tanto, depende de cada gestor de transacciones individuales y, por lo tanto, del servidor de aplicaciones, cómo se recuperan los recursos XA de una transacción XA. Es posible que en algunos servidores de aplicaciones, el adaptador de recursos de IBM MQ 9.0 no implemente los mecanismos específicos del servidor de aplicaciones que se utilizan para realizar la recuperación de transacciones XA.

Conexiones coincidentes en ManagedConnectionFactory

Un servidor de aplicaciones puede invocar el método matchManagedConnections en una instancia de ManagedConnectionFactory proporcionada por el adaptador de recursos de IBM MQ. Solo se devuelve ManagedConnection si el método encuentra una que coincide con los argumentos **`javax.security.auth.Subject`** y **`javax.resource.spi.ConnectionRequestInfo`** que se han pasado al servidor de conexiones.

Limitaciones del adaptador de recursos de IBM MQ

El adaptador de recursos de IBM MQ está soportado en todas las plataformas de IBM MQ. No obstante, cuando utiliza el adaptador de recursos de IBM MQ, algunas de las características de IBM MQ están limitadas o no están disponibles.

El adaptador de recursos de IBM MQ tiene las limitaciones siguientes:

- A partir de IBM MQ 8.0, el adaptador de recursos es un adaptador de recursos de Java Platform, Enterprise Edition 7 (Java EE 7) que proporciona funciones de JMS 2.0. Por lo tanto, se debe instalar el adaptador de recursos de la IBM MQ 8.0 o posterior en un servidor de aplicaciones certificados de Java EE 7 o posterior. Se puede conectar en modo de cliente o enlaces con cualquier gestor de colas en servicio.
- Cuando se ejecuta en el servidor de aplicaciones WebSphere Liberty no se da soporte a las IBM MQ classes for Java estabilizadas. En otros servidores de aplicaciones, no se recomienda el uso de las IBM MQ classes for Java. Consulte la IBM nota técnica Utilización de interfaces de WebSphere MQ Java en entornos J2EE/JEE para obtener detalles de las consideraciones de IBM MQ classes for Java en Java EE.
- Cuando se ejecutan en el servidor de aplicaciones WebSphere Liberty en z/OS, se debe utilizar la característica wmqJmsClient-2.0. El soporte de JCA genérico no es posible en z/OS.
- El adaptador de recursos de IBM MQ no da soporte a los programas de salida de canal escritos en lenguajes que no sean Java.
- Mientras se está ejecutando un servidor de aplicaciones, el valor de la propiedad sslFipsRequired debe ser true para todos los recursos JCA o false para todos los recursos JCA. Esto es obligatorio aunque los recursos JCA no se utilicen simultáneamente. Si la propiedad sslFipsRequired tiene distintos valores para distintos recursos JCA, IBM MQ emite el código de razón MQRC_UNSUPPORTED_CIPHER_SUITE, aunque no se utilice una conexión TLS.
- No puede especificar más de un almacén de claves para un servidor de aplicaciones. Si se realizan conexiones a más de un gestor de colas, todas las conexiones deben utilizar el mismo almacén de claves. Esta limitación no se aplica a WebSphere Application Server.
- Si utiliza una tabla de definiciones de canal de cliente (CCDT) con más de una definición de canal de conexión adecuada con el cliente, en caso de una anomalía, el adaptador de recursos podría seleccionar una definición de canal diferente y, por tanto, un gestor de colas diferente de la CCDT, lo que provocaría problemas en la recuperación de transacciones. El adaptador de recursos no efectúa ninguna acción para evitar la utilización de dicha configuración; es responsabilidad del usuario evitar configuraciones que puedan causar problemas en la recuperación de transacciones.
- La funcionalidad de reintento de conexión no está soportada para conexiones de salida cuando se ejecuta en un contenedor Java EE (EJB/Servlet). El reintento de conexión no está soportado en absoluto para JMS de salida cuando el adaptador se utiliza en un contexto de contenedor de JEE , independientemente de la configuración de la transacción o para un uso no transaccional.

- Como se ha definido en la sección 9.1.9 de la especificación Java EE Connector Architecture versión 1.7, no está soportada la reautenticación de las conexiones JMS. El archivo `ra.xml` del adaptador de recurso de IBM MQ debe tener la propiedad **`reauthentication-support`** establecida en `false`. Si el servidor de aplicaciones intenta reautenticar una conexión JMS, el adaptador de recursos de IBM MQ genera una excepción javax.resource.spi.SecurityException con el código de mensaje MQJCA1028.

Tareas relacionadas

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Referencia relacionada

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

WebSphere Application Server y el adaptador de recursos de IBM MQ

El adaptador de recursos IBM MQ es utilizado por aplicaciones que realizan la mensajería JMS con el proveedor de mensajería IBM MQ en WebSphere Application Server.

Importante: No utilice el adaptador de recursos de IBM MQ con WebSphere Application Server 6.0 o WebSphere Application Server 6.1.

WebSphere Application Server traditional 9.0 incluye una versión del adaptador de recursos de IBM MQ 9.0 . El adaptador de recursos IBM MQ 9.0 o posterior se puede desplegar en versiones anteriores de WebSphere Application Server, ya que estas versiones no están certificados por Java EE 7.

V 9.3.0 JM 3.0 V 9.3.0 WebSphere Application Server traditional no da soporte actualmente a Jakarta EE. Consulte Declaración de soporte del adaptador de recursos deIBM MQ.

Si desea utilizar una aplicación JMS para acceder a los recursos de un gestor de colas IBM MQ desde WebSphere Application Server, utilice el proveedor de mensajería IBM MQ en WebSphere Application Server. El proveedor de mensajería IBM MQ contiene una versión de IBM MQ classes for JMS. Para obtener más información, consulte la nota técnica Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ? (¿Qué versión del adaptador de recursos de WebSphere MQ se suministra con WebSphere Application Server?).

Importante: No incluya ninguno de los archivos JAR de IBM MQ classes for JMS o IBM MQ classes for Java en la aplicación. Hacerlo puede dar lugar a ClassCastExceptions y ser difícil de mantener.

Liberty y el adaptador de recursos de IBM MQ

El adaptador de recursos de IBM MQ se puede instalar en WebSphere Liberty utilizando una característica Liberty . La característica que utilice dependerá de la versión del adaptador de recursos que esté instalando. Como alternativa y sujeto a ciertas restricciones, puede instalar el adaptador de recursos utilizando el soporte genérico de Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

Restricciones generales al instalar el adaptador de recursos en Liberty

Las restricciones siguientes se aplican al adaptador de recursos cuando se utiliza la característica wmqJmsClient-1.1 o wmqJmsClient-2.0 y también cuando se utiliza el soporte genérico de JCA:

- IBM MQ classes for Java no se admite en Liberty. No se deben utilizar con la característica de mensajería IBM MQ Liberty ni con el soporte genérico de JCA. Para obtener más información, consulte Utilización de interfaces WebSphere MQ Java en entornos J2EE/JEE.
- El adaptador de recursos de IBM MQ tiene un tipo de transporte de BINDINGS_THEN_CLIENT. Este tipo de transporte no se admite dentro de la característica de mensajería IBM MQ Liberty.
- Antes de IBM MQ 9.0, la característica Advanced Message Security (AMS) no se incluía en la característica de mensajería IBM MQ Liberty. No obstante, AMS se admite con un adaptador de recursos IBM MQ 9.0 o posterior.

Nota: En las versiones de IBM MQ mayores que IBM MQ 9.0.0.6 y IBM MQ 9.1.0.1 debe utilizar la característica transportSecurity-1.0 en lugar de la característica ssl-1.0 .

Si desea ver más información, consulte:

Habilitación de la comunicación SSL en Liberty
Valores predeterminados SSL en Liberty
Transport Security 1.0

Restricciones al utilizar las características de Liberty

Con WebSphere Liberty 8.5.5 Fix Pack 2 a WebSphere Liberty 8.5.5 Fix Pack 5 inclusive, solo estaba disponible la característica wmqJmsClient-1.1 y solo se podía utilizar JMS 1.1 . WebSphere Liberty 8.5.5 Fix Pack 6 ha añadido la característica wmqJmsClient-2.0 para que se pueda utilizar JMS 2.0 .

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 . Para utilizar el adaptador de recursos de IBM MQ para Jakarta Messaging con Liberty, debe utilizar una versión de Liberty que dé soporte a Jakarta EE. Debe utilizar el adaptador de recursos para Jakarta Messaging con la característica Liberty generic messaging-3.0 .

La característica que debe utilizar depende de la versión del adaptador de recursos que esté utilizando:

- El adaptador de recursos IBM MQ 8.0.0 Fix Pack 3 y posterior a IBM MQ 8.0 se puede utilizar únicamente con la característica wmqJmsClient-2.0.
- El adaptador de recursos de IBM MQ 9.0 solo se puede utilizar con la característica wmqJmsClient-2.0.
- V 9.3.0 JM 3.0 V 9.3.0 La característica messaging-3.0 permite trabajar con adaptadores de recursos de Jakarta Messaging 3.0 .

Restricciones al utilizar el soporte de JCA genérico

Si utiliza el soporte genérico de JCA, se aplican las restricciones siguientes:

- Debe especificar el nivel de JMS al utilizar el soporte de JCA genérico. JMS 2.0 y JCA 1.7 solo se deben utilizar con los adaptadores de recursos IBM MQ 8.0.0 Fix Pack 3 y posteriores a IBM MQ 8.0.
- No es posible ejecutar el adaptador de recursos de IBM MQ en z/OS utilizando el soporte genérico de JCA. Para ejecutar el adaptador de recursos de IBM MQ en z/OS, se debe ejecutar con la característica wmqJmsClient-1.1 o wmqJmsClient-2.0.
- La ubicación del adaptador de recursos se especifica utilizando el elemento XML siguiente:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Importante: El valor de la etiqueta ID puede ser cualquiera EXCEPTO `wmqJms`. Si utiliza `wmqJms` como ID, Liberty no podrá cargar el adaptador de recursos correctamente. Esto se debe a que `wmqJms` es el ID que se utiliza internamente para hacer referencia a la característica específica para IBM MQ. De hecho, crea una excepción NullPointerException.

Los ejemplos siguientes muestran algunos fragmentos de código de un archivo `server.xml` al ejecutar JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

Consejo: Observe que se utilizan las características jca-1.7 y jms-2.0, pero no la característica wmqJmsClient-2.0.

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Consejo: Observe el uso de `mqJms` para el ID, que es el valor preferido. No utilice `wmqJms`.

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

Consejo: Anote la referencia classloaderProvideral adaptador de recursos a través del ID `mqJms`; esto es para permitir que se carguen las clases específicas de IBM MQ.

Restricciones al rastrear utilizando el soporte de JCA genérico

El rastreo y el registro no se integran dentro del sistema de rastreo de Liberty. En su lugar, el rastreo del adaptador de recursos de IBM MQ debe estar habilitado utilizando las propiedades del sistema Java o un archivo de configuración IBM MQ classes for JMS , tal como se describe en Rastreo de aplicaciones IBM MQ classes for JMS. Para obtener detalles sobre cómo establecer las propiedades del sistema Java en Liberty, consulte la documentación deWebSphere Liberty.

Por ejemplo, para habilitar el rastreo del adaptador de recursos IBM MQ en Liberty 19.0.0.9, añada una entrada al archivo Liberty `jvm.options`:

1. Cree un archivo de texto denominado `jvm.options`.
2. Inserte las siguientes opciones de JVM para habilitar el rastreo, una por línea, en este archivo:

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Para aplicar estos valores a un único servidor, guarde `jvm.options` en:

   ```
   ${server.config.dir}/jvm.options
   ```

   Para aplicar estos cambios a todos los Liberty, guarde `jvm.options` en:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Esto entrará en vigor para todas las JVM que no tengan un archivo `jvm.options` definido localmente.
4. Reinicie el servidor para habilitar los cambios.

Esto hace que el rastreo se grabe en un archivo de rastreo denominado `MQRA-WLP_<process identifier>.trc` en el directorio `<path_to_trace_to>`.

Soporte completo de Liberty XA con tablas de definición de canal de cliente

Al utilizar WebSphere Liberty 18.0.0.2 en adelante, con IBM MQ 9.2.0 o posterior, puede utilizar grupos de gestores de colas dentro de la tabla de definiciones de canal de cliente (CCDT) junto con transacciones XA. Esto significa que ahora se puede utilizar la distribución de la carga de trabajo y la disponibilidad que proporcionan los grupos de gestores de colas manteniendo, al mismo tiempo, la integridad de las transacciones.

En el supuesto de errores de conectividad con un gestor de colas, el gestor de colas debe volver a estar disponible para que la transacción se pueda resolver. La recuperación de las transacciones se gestiona mediante Liberty, y es posible que tenga que configurar el gestor de transacciones, de modo que se permita un periodo de tiempo apropiado para que los gestores de colas vuelvan a estar disponibles.
Si desea más información, consulte Gestor de transacciones (transacción) en la documentación del producto WebSphere Liberty.

Se trata de una característica del lado del cliente, es decir, necesita un adaptador de recursos IBM MQ 9.2.0 o posterior, no un gestor de colas IBM MQ 9.2.0 o posterior.

Instalación del adaptador de recursos de IBM MQ

El adaptador de recursos IBM MQ se proporciona como un archivo de archivado de recursos (RAR). Instale el archivo RAR en el servidor de aplicaciones. Puede ser necesario añadir directorios a la vía de acceso del sistema.

Acerca de esta tarea

El adaptador de recursos de IBM MQ se proporciona como un archivo de archivado de recursos (RAR):

- V 9.3.0 JM 3.0 V 9.3.0 Para Jakarta Messaging 3.0, este archivo se denomina `wmq.jakarta.jmsra.rar`. El archivo RAR contiene IBM MQ classes for Jakarta Messaging y la implementación de IBM MQ de las interfaces Jakarta Connectors Architecture (JCA).
- JMS 2.0 Para JMS 2.0, este archivo se denomina `wmq.jmsra.rar`. El archivo RAR contiene IBM MQ classes for JMS y la implementación de IBM MQ de las interfaces Java EE Connector Architecture (JCA).

Cuando instala el adaptador de recursos como parte de la instalación del producto IBM MQ , el archivo RAR se instala con IBM MQ classes for JMS en el directorio que se muestra en la Tabla 61 en la página 457.

Tabla 61. Directorio IBM MQ que contiene el archivo RAR para cada plataforma

Plataforma	**Directorio**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Debe utilizar el adaptador de recursos de IBM MQ para conectar con IBM MQ desde un servidor de aplicaciones. Dependiendo del servidor de aplicaciones que esté utilizando, el adaptador de recursos puede estar preinstalado o ser necesario instalarlo.

Tabla 62. Instalación del adaptador de recursos en un servidor de aplicaciones

Servidor de aplicaciones	**¿Preinstalado o es necesario instalarlo?**
WebSphere Application Server traditional 9.0	El adaptador de recursos de IBM MQ 9.0 está instalado previamente en WebSphere Application Server traditional 9.0. Por lo tanto, no es necesario instalar un nuevo adaptador de recursos en WebSphere Application Server traditional 9.0.
WebSphere Liberty	WebSphere Liberty no contiene el adaptador de recursos de IBM MQ, por lo que debe obtenerlo por separado desde Fix Central.
Otro servidor de aplicaciones Java EE o Jakarta EE	Obtenga el adaptador de recursos por separado de Fix Central, como para WebSphere Liberty.

Procedimiento

- Si se está conectando a IBM MQ desde WebSphere Liberty, u otro servidor de aplicaciones Java EE o Jakarta EE , descargue e instale el adaptador de recursos IBM MQ tal como se describe en "Instalación del adaptador de recursos en Liberty" en la página 458.
- Linux AIX
  Si utiliza conexiones de enlaces en sistemas AIX and Linux, asegúrese de que el directorio donde residen las bibliotecas de JNI (Java Native Interface) esté en la vía de acceso del sistema.

  Conocer la ubicación de este directorio, que también contiene las bibliotecas de IBM MQ classes for JMS, consulte "Configuración de las bibliotecas JNI (Java Native Interface)" en la página 100.

  Windows En Windows, este directorio se añade automáticamente a la vía de acceso del sistema durante la instalación de IBM MQ classes for JMS.

  Consejo: Como alternativa a establecer la vía de acceso del sistema, el adaptador de recursos de IBM MQ tiene una propiedad denominada nativeLibraryPath que se puede utilizar para especificar la ubicación de la biblioteca de JNI. Por ejemplo, en WebSphere Liberty , esto se configuraría tal como se muestra en el ejemplo siguiente:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  Las transacciones están soportadas tanto en la modalidad de cliente como en la modalidad de enlaces.

Instalación del adaptador de recursos en Liberty

Para conectarse a IBM MQ desde WebSphere Liberty, u otros servidores de aplicaciones Java EE o Jakarta EE , debe utilizar el adaptador de recursos IBM MQ . Puesto que Liberty no contiene el adaptador de recursos IBM MQ, debe obtenerlo por separado del Fix Central.

Antes de empezar

Nota: La información de este tema no se aplica a WebSphere Application Server traditional 9.0. El adaptador de recursos de IBM MQ 9.0 está preinstalado en WebSphere Application Server traditional 9.0. Por lo tanto, no es necesario instalar un adaptador de recursos nuevo en este caso.

Antes de iniciar esta tarea, asegúrese de que tiene instalado un Java runtime environment (JRE) en la máquina y que el JRE se ha añadido a la vía de acceso del sistema.

El instalador de Java que se utiliza en este proceso de instalación no requiere que se ejecute como usuario root ni ningún otro específico. El único requisito es que el usuario que se ejecuta tenga acceso de escritura en el directorio al que desea que se vayan los archivos.

Para las versiones de Liberty hasta WebSphere Liberty 8.5.5 Fix Pack 1, si se despliega un EJB utilizando solo la configuración dentro de `ejb-jar.xml`, la versión de WebSphere Application Server que está utilizando el perfil Liberty debe haber aplicado el APAR PM89890. Este método de configuración se utiliza para el programa de verificación de instalación (IVT) del adaptador de recursos, de modo que este APAR es necesario para que la IVT se ejecute.

V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, se da soporte a Jakarta Messaging 3.0 . Para utilizar el adaptador de recursos de IBM MQ para Jakarta Messaging con Liberty, debe utilizar una versión de Liberty que dé soporte a Jakarta EE. Por ejemplo, puede utilizar la característica Liberty generic messaging-3.0 .

Acerca de esta tarea

El archivo JAR para el adaptador de recursos que puede descargar desde Fix Central es ejecutable. Cuando se ejecuta este archivo, se muestra el acuerdo de licencia de IBM MQ, que debe aceptarse. Se solicita un directorio en el que instalar el adaptador de recursos de IBM MQ. El archivo RAR de adaptador de recursos y el programa de prueba de verificación de instalación (IVT) se instalan en ese

directorio. Puede aceptar el valor predeterminado o especificar otro directorio, que puede ser el directorio de adaptadores de recursos de un servidor de aplicaciones, o cualquier otro directorio del sistema. El directorio se crea como parte de la instalación si no existe.

Antes de IBM MQ 9.0, el nombre el archivo a descargar estaba en formato *V.R.M.F*-WS-MQ-Java-InstallRA.jar, por ejemplo, 8.0.0.6-WS-MQ-Java-InstallRA.jar. A partir de IBM MQ 9.0, el formato del nombre de archivo es *V.R.M.F*-IBM-MQ-Java-InstallRA.jar, por ejemplo 9.0.0.0-IBM-MQ-Java-InstallRA.jar.

Una vez que haya descargado e instalado el adaptador de recursos, estará preparado para configurarlo en WebSphere Liberty.

Procedimiento

1. Descargue el adaptador de recursos IBM MQ desde Fix Central.
   a) Pulse este enlace: Adaptador de recursos IBM MQ.
   b) Busque el adaptador de recursos para la versión de IBM MQ en la lista de arreglos disponibles que aparece.

      Por ejemplo:

      ```
      release level: 9.1.4.0-IBM-MQ-Java-InstallRA
      Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
      Servers
      ```

      A continuación, pulse el nombre de archivo del adaptador de recursos y siga el proceso de descarga.
2. Inicie la instalación especificando el mandato siguiente desde el directorio en el que ha descargado el archivo.

   A partir de IBM MQ 9.0, el formato del mandato es el siguiente:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   donde *V.R.M.F* es la versión, el release, la modificación y el número de fixpack y *V.R.M.F*-IBM-MQ-Java-InstallRA.jar es el nombre del archivo que se ha descargado desde Fix Central.

   Por ejemplo, para instalar el adaptador de recursos de IBM MQ para el release de IBM MQ 9.1.4 , utilizaría el mandato siguiente:

   ```
   java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
   ```

   Nota: Para realizar esta instalación, debe tener instalado un JRE en su equipo y añadido a la vía de acceso del sistema.

   Cuando se especifica el mandato, se muestra la siguiente información:

   ```
   Para poder utilizar, extraer o instalar IBM MQ 9.1, debe aceptar
   los términos de 1. IBM Acuerdo Internacional de Licencia para la Evaluación de
   Programas 2. IBM y
   información adicional de licencia. Lea detenidamente los siguientes acuerdos de licencia.

   El acuerdo de licencia se puede ver por separado utilizando la
   opción --viewLicenseAgreement.
   Pulse Intro para mostrar ahora los términos de la licencia o 'x' para omitir.
   ```
3. Revise y acepte los términos de la licencia:
   a) Para ver la licencia, pulse Intro.

      Si se pulsa x, se omite la visualización de la licencia.

      Después de la visualización de la licencia o inmediatamente después de seleccionar x, aparece el mensaje siguiente para indicar que puede elegir la visualización de términos de licencia adicionales:

```
La información de licencia adicional se puede ver por separado utilizando la
opción --viewLicenseInfo.
Pulse Intro para mostrar ahora la información de la licencia adicional o 'x' para omitir.
```

b) Para ver los términos de licencia adicionales, pulse Intro.

Si se pulsa x, se omite la visualización de los términos adicionales de la licencia.

Tras la visualización de los términos de licencia adicionales o inmediatamente después de seleccionar x, aparece el siguiente mensaje solicitando que acepte el acuerdo de licencia:

```
Al elegir la opción de "Acepto" a continuación, acepta los términos del
acuerdo de licencia y los términos que no son de IBM, si procede. Si no
está de acuerdo, seleccione "No acepto".

Seleccione [1] Acepto, o [2] No acepto:
```

c) Para aceptar el acuerdo de licencia y continuar con la selección del directorio de instalación, seleccione 1.

Por el contrario, si selecciona 2, la instalación termina de forma inmediata.

Si ha seleccionado 1, aparecerá el siguiente mensaje, pidiéndole que seleccione un directorio de instalación de destino:

```
Especifique el directorio para los archivos de producto o déjelo en blanco para aceptar los
valores predeterminados.
El directorio de destino predeterminado es H:\Liberty\WMQ
¿Directorio de destino para los archivos del producto?
```

4. Especifique el directorio de instalación para el adaptador de recursos:

- Si desea instalar el adaptador de recursos en la ubicación predeterminada, pulse Intro sin especificar un valor.
- Si desea instalar el adaptador de recursos en una ubicación distinta de la predeterminada, especifique el nombre del directorio en el que desea instalar el adaptador de recursos y, a continuación, pulse Intro.

Después de haber instalado los archivos en la ubicación seleccionada, aparece un mensaje de confirmación, tal como se muestra en el ejemplo siguiente:

```
Extrayendo archivos en H:\Liberty\WMQ\wmq
Se han extraído satisfactoriamente todos los archivos del producto.
```

Durante la instalación, se crea un nuevo directorio con el nombre `wmq` dentro del directorio de instalación seleccionado y, a continuación, se instalaron los archivos siguientes en el directorio `wmq`:

- El programa de prueba de verificación de la instalación, `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) o `wmq.jmsra.ivt` (JMS 2.0).
- El archivo RAR de IBM MQ ,`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 o `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

Opcional: Configure el adaptador de recursos Java EE 7 (JMS 2.0) en WebSphere Liberty Profile.

Los pasos que debe llevar a cabo para configurar el adaptador de recursos en Liberty son los siguientes. Para obtener más información, consulte la documentación del producto WebSphere Application Server.

a) Añada la característica wmqJmsClient-2.0 en el archivo `server.xml` para permitir trabajar con el adaptador de recursos de IBM MQ.

Para obtener más información, consulte “Qué versión del adaptador de recursos se debe utilizar” en la página 449.

b) Añada una referencia al archivo `wmq.jmsra.rar` (JMS 2.0) que ha instalado.

A continuación, se muestra un ejemplo de configuración para dar soporte a servlets y MDB, con JNDI:

```
<featureManager>
   <feature>wmqJmsClient-2.0</feature>
   <feature>servlet-3.0</feature>
   <feature>jmsMdb-3.1</feature>
   <feature>jndi-1.0</feature>
   </featureManager>
```

```
<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

6. JM 3.0

Opcional: Configure el adaptador de recursos Jakarta EE 9 (Jakarta Messaging 3.0) en WebSphere Liberty Profile.

Los pasos que debe llevar a cabo para configurar el adaptador de recursos en Liberty son los siguientes. Para obtener más información, consulte la documentación del producto WebSphere Application Server.

a) Añada la característica wmqJmsClient-3.0 al archivo `server.xml` para permitir trabajar con el adaptador de recursos IBM MQ .

Para obtener más información, consulte “Qué versión del adaptador de recursos se debe utilizar” en la página 449.

b) Añada una referencia al archivo `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) que ha instalado.

A continuación, se muestra un ejemplo de configuración para dar soporte a servlets y MDB, con JNDI:

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

Nota: Si utiliza Open Liberty, en lugar de WebSphere Liberty Profile, tendrá que utilizar la característica de soporte de adaptador de recursos genérico "messagingClient-3.0" en lugar de "wmqJmsClient-3.0" y otros aspectos de la configuración serán diferentes. Consulte la documentación de Open Liberty para obtener más detalles.

Configuración del adaptador de recursos de IBM MQ

Para configurar el adaptador de recursos de IBM MQ, debe definir diversos recursos de Java Platform, Enterprise Edition Connector Architecture (JCA) y opcionalmente propiedades del sistema. Debe también configurar el adaptador de recursos para que ejecute el programa de prueba de verificación de la instalación (IVT). Esto es importante porque el servicio técnico de IBM puede solicitar la ejecución de ese programa para comprobar que cualquier servidor de aplicaciones que no sea de IBM se haya configurado correctamente.

Antes de empezar

En esta tarea se supone que el usuario ya está familiarizado con JMS e IBM MQ classes for JMS. Muchas de las propiedades utilizadas para configurar el adaptador de recursos de IBM MQ son equivalentes a propiedades de objetos de IBM MQ classes for JMS y tienen la misma función.

Acerca de esta tarea

Cada servidor de aplicaciones proporciona su propio conjunto de interfaces de administración. Algunos servidores de aplicaciones proporcionan interfaces gráficas de usuario para definir recursos de JCA, pero otros necesitan que el administrador escriba planes de despliegue XML. Por tanto, queda fuera del ámbito de esta publicación el proporcionar información sobre cómo configurar el adaptador de recursos de IBM MQ para cada servidor de aplicaciones.

Los pasos siguientes se centran pues solamente en lo que se necesita configurar. Consulte la documentación proporcionada con su servidor de aplicaciones para conocer cómo configurar un adaptador de recursos de JCA.

Procedimiento

Defina recursos de JCA en las categorías siguientes:

- Defina las propiedades del objeto ResourceAdapter.

  Estas propiedades, que representan las propiedades globales del adaptador de recursos, tales como el nivel de rastreo de diagnóstico, se describen en "Configuración de las propiedades del objeto ResourceAdapter" en la página 463.
- Defina las propiedades de un objeto ActivationSpec.

  Estas propiedades determinan cómo se activa un MDB para la comunicación de entrada. Para obtener más información, consulte "Configuración del adaptador de recursos para la comunicación de entrada" en la página 465.
- Defina las propiedades de un objeto ConnectionFactory.

  El servidor de aplicaciones utiliza estas propiedades para crear un objeto ConnectionFactory de JMS para la comunicación de salida. Para obtener más información, consulte "Configuración del adaptador de recursos para la comunicación de salida" en la página 485.
- Defina las propiedades de un objeto de destino administrado.

  El servidor de aplicaciones utiliza estas propiedades para crear un objeto Queue de JMS o un objeto Topic de JMS para la comunicación de salida. Para obtener más información, consulte "Configuración del adaptador de recursos para la comunicación de salida" en la página 485.
- Opcional: Defina un plan de despliegue para el adaptador de recursos.

  El archivo RAR del adaptador de recursos de IBM MQ contiene un archivo llamado `META-INF/ra.xml` que contiene un descriptor de despliegue para el adaptador de recursos. Este descriptor de despliegue está definido por el esquema XML en https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd y contiene información sobre el adaptador de recursos y los servicios que proporciona. Un servidor de aplicaciones puede también necesitar un plan de despliegue para el adaptador de recursos. Este plan de despliegue es específico del servidor de aplicaciones.

Especifique las propiedades del sistema JVM según sea necesario:

- Si utiliza TLS (Transport Layer Security), especifique las ubicaciones del archivo de almacén de claves y del archivo de almacén de confianza como propiedades de sistema JVM, como en el ejemplo siguiente:

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  Estas propiedades no pueden ser propiedades de un objeto ActivationSpec ni ConnectionFactory, y no puede especificar más de un almacén de claves para un servidor de aplicaciones. Las propiedades se aplican a la JVM completa y, por tanto, pueden afectar al servidor de aplicaciones si otras aplicaciones, que se ejecutan en el servidor de aplicaciones, utilizan conexiones TLS. También es posible que el servidor de aplicaciones restablezca estas propiedades en valores distintos. Para obtener más información sobre el uso de TLS con IBM MQ classes for JMS, consulte "Utilización de TLS con IBM MQ classes for JMS" en la página 261.
- Opcional: Si es necesario, configure el adaptador de recursos para registrar mensajes de aviso en el registro de salida estándar del servidor de aplicaciones.

  Los archivos de registro, mensajes de aviso y mensajes de error del adaptador de recursos utilizan el mismo mecanismo que IBM MQ classes for JMS. Para obtener más información, consulte Registro de errores para IBM MQ classes for JMS. Esto significa que, de forma predeterminada, los mensajes se escriben en un archivo llamado `mqjms.log`. Para configurar el adaptador de recursos para registrar adicionalmente mensajes de aviso en el registro de salida estándar del servidor de aplicaciones, establezca la siguiente propiedad del sistema JVM para el servidor de aplicaciones:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

Esta es la misma propiedad que la que se utiliza para controlar el rastreo de IBM MQ classes for JMS. Al igual que ocurre con IBM MQ classes for JMS, es posible utilizar una propiedad del sistema que apunte al archivo `jms.config` (consulte "El archivo de configuración IBM MQ classes for JMS/Jakarta Messaging" en la página 102). Para obtener información sobre cómo establecer una propiedad del sistema JVM, consulte la documentación del servidor de aplicaciones.

Configure el adaptador de recursos para ejecutar la prueba de verificación de la instalación

- Configure el adaptador de recursos para ejecutar el programa de prueba de verificación de la instalación (IVT) que se proporciona con el adaptador de recursos de IBM MQ.

  Para obtener más información sobre lo que debe configurar para poder ejecutar el programa IVT, consulte "Verificación de la instalación del adaptador de recursos" en la página 507.

  Esto es importante porque el servicio técnico de IBM puede solicitar la ejecución de ese programa para comprobar que cualquier servidor de aplicaciones que no sea de IBM se haya configurado correctamente.

  Importante: Debe configurar el adaptador de recursos antes de poder ejecutar el programa.

Configuración de las propiedades del objeto ResourceAdapter

El objeto ResourceAdapter encapsula las propiedades globales del adaptador de recursos de IBM MQ, tales como el nivel de rastreo de diagnóstico. Para definir estas propiedades, utilice las funciones del adaptador de recursos, tal como se describe en la documentación proporcionada con el servidor de aplicaciones.

El objeto ResourceAdapter tiene dos conjuntos de propiedades:

- Propiedades asociadas con el rastreo de diagnóstico
- Propiedades asociadas con la agrupación de conexiones gestionada por el adaptador de recursos

La forma en la que se definen estas propiedades depende de las interfaces de administración que proporciona el servidor de aplicaciones. Si está utilizando WebSphere Application Server traditional, consulte "Configuración de WebSphere Application Server traditional" en la página 465 o si está utilizando WebSphere Liberty, consulte "Configuración de WebSphere Liberty" en la página 465. Para otros servidores de aplicaciones, consulte la documentación del producto correspondiente al servidor de aplicaciones.

Para obtener más información sobre cómo definir las propiedades asociadas al rastreo de diagnóstico, consulte Rastreo del adaptador de recursos IBM MQ

El adaptador de recursos gestiona una agrupación de conexiones internas de las conexiones de JMS que se utilizan para entregar mensajes a los beans controlados por mensajes (MDB). La Tabla 63 en la página 463 lista las propiedades del objeto ResourceAdapter que están asociadas con la agrupación de conexiones.

Tabla 63. Propiedades del objeto ResourceAdapter que están asociadas con la agrupación de conexiones

Nombre de la propiedad	Tipo	Valor predeterminado	Descripción
maxConnections	Serie	50	Número máximo de conexiones con un gestor de colas de IBM MQ y número máximo de beans controlados por mensajes desplegados.
connectionConcurrency	Serie	1	Número máximo de beans controlados por mensajes para compartir una conexión de JMS. El uso compartido de conexiones no es posible y el valor de esta propiedad es siempre 1.

Tabla 63. Propiedades del objeto ResourceAdapter que están asociadas con la agrupación de conexiones (continuación)

Nombre de la propiedad	Tipo	Valor predeterminado	Descripción
reconnectionRetryCount	Serie	5	Número máximo de intentos realizados por el adaptador de recursos para volver a conectar con un gestor de colas de IBM MQ si falla una conexión.
reconnectionRetryInterval	Serie	300000	Tiempo, en milisegundos, que el adaptador de recursos espera antes de intentar volver a conectar con un gestor de colas de IBM MQ.
startupRetryCount	Serie	0	Número predeterminado de veces que se debe intentar conectar con un bean controlado por mensajes en el arranque si el gestor de colas no está en ejecución cuando se inicia el servidor de aplicaciones.
startupRetryInterval	Serie	30000	Tiempo de inactividad predeterminado entre los intentos de conexión durante el arranque (en milisegundos).
supportMQExtensions	Serie	falso	Revierte el comportamiento de IBM MQ JMS al comportamiento anterior a JMS 2.0. Para obtener más información, consulte “Propiedad SupportMQExtensions” en la página 337.
nativeLibraryVía de acceso	Serie	<empty>	Vía de acceso que se debe utilizar para cargar la biblioteca JNI de IBM MQ a fin de permitir las conexiones en la modalidad de enlaces. Windows En Windows, la vía de acceso del sistema también necesita contener la ubicación de la correspondiente instalación de IBM MQ.

Cuando se despliega un bean controlado por mensaje en el servidor de aplicaciones, se crea una nueva conexión de JMS y se inicia una conversación con el gestor de colas, siempre que no se sobrepase el número máximo de conexiones especificado por la propiedad maxConnection. Por lo tanto, el número máximo de beans controlados por mensaje es igual al número máximo de conexiones. Si el número de beans desplegados alcanza este máximo, cualquier intento de desplegar otro bean fallará. Si se detiene un bean, otro bean puede utilizar su conexión.

En general, si se deben desplegar muchos beans controlados por mensaje, debe aumentar el valor de la propiedad maxConnections.

Las propiedades reconnectionRetryCount y reconnectionRetryInterval controlan el comportamiento del adaptador de recursos cuando falla la conexión con un gestor de colas de IBM MQ, debido a un error de red, por ejemplo. Cuando falla una conexión, el adaptador de recursos suspende la entrega de mensajes a todos los beans proporcionados por esa conexión durante el intervalo especificado por la propiedad reconnectionRetryInterval. A continuación, el adaptador de recursos intenta volver a conectar con el gestor de colas. Si el intento falla, el adaptador de recursos realiza más intentos de reconexión de acuerdo con los intervalos de tiempo especificados por la propiedad reconnectionRetryInterval, hasta que se alcanza el límite establecido por la propiedad reconnectionRetryCount. Si fallan todos los intentos, la entrega se detiene permanentemente hasta que los beans se reinicien manualmente.

En general, el objeto ResourceAdapter no necesita administración. Pero para habilitar el rastreo de diagnósticos en sistemas AIX and Linux, por ejemplo, puede establecer las propiedades siguientes:

```
traceEnabled:    true
traceLevel:      10
```

Estas propiedades no tienen ningún efecto si el adaptador de recursos no se ha iniciado, que es el caso, por ejemplo, cuando las aplicaciones que utilizan recursos de IBM MQ sólo se ejecutan en el contenedor de cliente. En esta situación, puede establecer las propiedades del rastreo de diagnóstico como propiedades del sistema de la máquina virtual Java (JVM). Puede establecer las propiedades utilizando el distintivo **-D** en el mandato **java**, como en el siguiente ejemplo:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

No es necesario que defina todas las propiedades del objeto ResourceAdapter. Las propiedades que no se especifican toman sus valores predeterminados. En un entorno gestionado, es mejor no mezclar las dos maneras de especificar propiedades. Si las mezcla, las propiedades del sistema JVM tienen prioridad sobre las propiedades del objeto ResourceAdapter.

Configuración de WebSphere Application Server traditional

Las mismas propiedades están disponibles para el adaptador de recursos en WebSphere Application Server traditional, pero se deben establecer dentro del panel de propiedades del adaptador de recursos (consulte Valores del proveedor JMS en la documentación del producto de WebSphere Application Server traditional. El rastreo está controlado por la sección de diagnósticos de la configuración de WebSphere Application Server traditional. Para obtener más información, consulte Utilización de proveedores de diagnósticos en la documentación del producto de WebSphere Application Server traditional.

Configuración de WebSphere Liberty

El adaptador de recursos se configura utilizando elementos XML en el archivo `server.xml`, tal como se muestra en el ejemplo siguiente:

JMS 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

El rastreo se habilita añadiendo este elemento XML:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

Configuración del adaptador de recursos para la comunicación de entrada

Para configurar la comunicación de entrada, defina las propiedades de uno o más objetos ActivationSpec.

Las propiedades de un objeto ActivationSpec determinan cómo un bean controlado por mensajes (MDB) recibe JMS mensajes de una cola IBM MQ . El comportamiento transaccional del MDB está definido en su descriptor de despliegue.

Un objeto ActivationSpec tiene dos conjuntos de propiedades:

- Propiedades que se utilizan para crear una conexión JMS con un gestor de colas de IBM MQ
- Propiedades que se utilizan para crear un consumidor de conexión de JMS que entrega mensajes asíncronamente a medida que llegan en una cola especificada

La manera en que define las propiedades de un objeto ActivationSpec depende de las interfaces de administración proporcionadas por el servidor de aplicaciones.

Propiedades utilizadas para crear una conexión JMS con un gestor de colas de IBM MQ

Todas las propiedades contenidas en la Tabla 64 en la página 466 son opcionales.

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
applicationName	Serie	• Nombre de la clase que realiza la llamada, si está disponible, ajustada para que no sea más larga de 28 caracteres. Si no está disponible, se utiliza la serie `WebSphere MQ Client for Java`.	Nombre con el que se registra una aplicación en el gestor de colas. Este nombre de aplicación se muestra mediante el mandato **`DISPLAY CONN MQSC/PCF`** (donde el campo se denomina **`APPLTAG`**) o en la pantalla IBM MQ Explorer **Conexiones de aplicación** (donde el campo se denomina **`App name`**).
brokerCCDurSubQueue [1]	Serie	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • El nombre de una cola	El nombre de la cola de la que un consumidor de conexión recibe mensajes de suscripciones duraderas
brokerCCSubQueue [1]	Serie	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • El nombre de una cola	El nombre de la cola de la que un consumidor de conexión recibe mensajes de suscripciones no duraderas
brokerControlQueue [1]	Serie	• **SYSTEM.BROKER.CONTROL.QUEUE** • El nombre de una cola	El nombre de la cola de control de intermediario
brokerQueueManager [1]	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	Nombre del gestor de colas en el que se ejecuta el intermediario
brokerSubQueue [1]	Serie	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • El nombre de una cola	El nombre de la cola de la que un consumidor de mensajes no duraderos recibe mensajes

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
brokerVersion [1]	Serie	• **sin especificar** - Después de migrar el intermediario desde la Versión 6 a la Versión 7, establezca esta propiedad para que ya no se utilicen cabeceras RFH2. Después de la migración, esta propiedad deja de ser relevante. • **V1** - Para utilizar un intermediario de publicación/suscripción de IBM MQ. Este valor es el valor predeterminado si TRANSPORT se establece en BIND o CLIENT. • **V2** - Para utilizar un intermediario de IBM Integration Bus en modalidad nativa. Este valor es el valor predeterminado si TRANSPORT se establece en DIRECT o DIRECTHTTP.	Versión del intermediario que se utiliza
ccdtURL	Serie	• **null** • Un localizador universal de recursos (URL)	URL que identifica el nombre y la ubicación del archivo que contiene la tabla de definición de canal de cliente (CCDT) y especifica cómo se puede acceder al archivo
CCSID	Serie	• **819** • Identificador de juego de caracteres codificados soportado por la máquina virtual Java (JVM)	El identificador de juego de caracteres codificado para una conexión
canal	Serie	• **SYSTEM.DEF.SVRCONN** • El nombre de un canal MQI	El nombre del canal MQI que se va a utilizar
cleanupInterval [1]	int	• **3 600 000** • Un entero positivo	El intervalo, en milisegundos, entre ejecuciones de fondo del programa de utilidad de limpieza de publicación/suscripción
cleanupLevel [1]	Serie	• **SAFE** • NINGUNO • STRONG • FORCE • NONDUR	Nivel de limpieza para un almacén de suscripción basado en intermediario
clientID	Serie	• **null** • Un identificador de cliente	El identificador de cliente para una conexión

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
cloneSupport	Serie	• **DISABLED** - Sólo se puede ejecutar una instancia de un suscriptor de temas duradero simultáneamente. • ENABLED - Se pueden ejecutar simultáneamente dos o más instancias del mismo suscriptor de tema duradero, pero cada instancia se debe ejecutar en una máquina virtual Java (JVM) separada.	Determina si dos o más instancias del mismo suscriptor de temas duradero pueden ejecutarse simultáneamente
connectionFactoryLookup	Serie	• **null** • El nombre JNDI de un objeto ConnectionFactory	Si se establece esta propiedad, ActivationSpec busca un objeto ConnectionFactory de JMS que tenga el nombre JNDI especificado en el espacio de nombres JNDI del servidor de aplicaciones y luego utiliza las propiedades de ese objeto para crear una conexión JMS con un gestor de colas de IBM MQ, con una excepción. La única propiedad de ActivationSpec que se utilizará al crear la conexión JMS es clientID. Para obtener más información, consulte “Propiedades connectionFactoryLookup y destinationLookup de ActivationSpec” en la página 481.

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
connectionNameList	Serie	• **localhost(1414)** • Serie compuesta por elementos separados por comas donde cada elemento tiene el formato: `HOSTNAME(PORT)` donde *NOMBRE_HOST* es un nombre DNS o una dirección IP.	Lista de nombres de conexión TCP/IP utilizados para las comunicaciones de entrada. Cuando se especifica, **connectionNameList** reemplaza a las propiedades **hostname** y **port**. Esta propiedad se utiliza para volver a conectar con gestores de colas de varias instancias. **connectionNameList** es similar en forma a **localAddress**, pero no se debe confundir con él. **localAddress** especifica las características de las comunicaciones locales, mientras que **connectionNameList** especifica cómo acceder a un gestor de colas remoto.
V 9.3.0 V 9.3.0 dynamicallyBalanced [4]	Boolean	• **false** • true	Indica si se puede solicitar a este MDB que reciba mensajes de un gestor de colas diferente como parte del equilibrio de aplicaciones en un clúster uniforme.
failIfQuiesce	Boolean	• **true** • falso	Determina si las llamadas a determinados métodos no responden si el gestor de colas está en un estado inmovilizado.
headerCompression	Serie	• **NONE** • SYSTEM - Se realiza compresión de cabecera de mensaje RLE	Lista de las técnicas que se pueden utilizar para comprimir datos de cabecera en una conexión

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
hostName	Serie	• **localhost** • El nombre de un host • Una dirección IP	Nombre de host o dirección IP del sistema en el que reside el gestor de colas. Cuando está especificada, la propiedad **connectionNameList** reemplaza a las propiedades **hostname** y **port**.
localAddress	Serie	• **null** • Una serie con el formato: `[ host_name ][(low_port [, high_port ])]` donde *nombre_host* es el nombre de un host o una dirección IP, *puerto_inferior* y *puerto_superior* son números de puertos TCP, y los delimitadores indican un componente opcional	Para una conexión con un gestor de colas, esta propiedad especifica uno o ambos elementos siguientes: • La interfaz de red local que se utilizará • El puerto local o un rango de puertos locales que se utilizará **localAddress** es similar en forma a **connectionNameList**, pero no se debe confundir con él. **localAddress** especifica las características de las comunicaciones locales, mientras que **connectionNameList** especifica cómo acceder a un gestor de colas remoto.
messageCompression	Serie	• **NONE** • Una lista de uno o varios de los valores siguientes separados por caracteres en blanco: RLE ZLIBFAST ZLIBHIGH	Lista de las técnicas que se pueden utilizar para comprimir datos de mensaje en una conexión
messageRetention [1]	Boolean	• **true** - Los mensajes no deseados permanecen en la cola de entrada. • false - Los mensajes no deseados se tratan conforme a sus opciones de disposición.	Determina si el consumidor de conexión mantiene los mensajes que no se desean en la cola de entrada.

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
messageSelection [1]	Serie	• **CLIENT** • BROKER	Determina si la selección de mensajes es realizada por IBM MQ classes for JMS o por el intermediario. La selección de mensajes por el intermediario no está soportada cuando brokerVersion tiene el valor 1.
Contraseña	Serie	• **null** • Una contraseña	La contraseña predeterminada a utilizar cuando se crea una conexión con el gestor de colas
pollingInterval [1]	int	• **5000** • Cualquier entero positivo	Si cada escucha de mensajes en una sesión no tiene un mensaje adecuado en cola, este valor es el intervalo máximo, en milisegundos, que transcurre antes de que cada escucha de mensajes intente de nuevo obtener un mensaje de su cola. Si con frecuencia sucede esto de que no haya disponible ningún mensaje adecuado para ninguno de los escuchas de mensajes de una sesión, considere aumentar el valor de esta propiedad. Esta propiedad sólo tiene relevancia si TRANSPORT tiene el valor BIND o CLIENT.
port	int	• **1414** • Un número de puerto TCP	El puerto en el que el gestor de colas está a la escucha. Cuando está especificada, la propiedad **connectionNameList** reemplaza a las propiedades **hostname** y **port**.

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
providerVersion	serie	• **sin especificar** • Serie de caracteres en uno de los formatos siguientes – V.R.M.F – V.R.M – V.R – V donde V, R, M y F son valores enteros mayores o iguales que cero.	Versión, release, nivel de modificación y fixpack del gestor de colas al que se debe conectar el MDB (bean controlado por mensaje).
queueManager	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	El nombre del gestor de colas al que se va a conectar
receiveExit [3]	Serie	• **null** • Serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa la interfaz MQReceiveExit de IBM MQ classes for Java.	Identifica un programa de salida de recepción de canal o una secuencia de programas de salida de recepción que se deben ejecutar sucesivamente
receiveExitInit	Serie	• **null** • Una serie que comprende uno o varios elementos de datos de usuario separados por comas	Los datos de usuario que se pasan a los programas de salidas de recepción de canal, cuando se les invoca

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
rescanInterval [1]	int	• **5000** • Cualquier entero positivo	Cuando un consumidor de mensajes en el dominio punto a punto utiliza un selector de mensajes para seleccionar qué mensajes desea recibir, IBM MQ classes for JMS busca en la cola IBM MQ los mensajes adecuados en la secuencia determinada por el atributo **MsgDeliverySequence** de la cola. Cuando IBM MQ classes for JMS encuentra un mensaje adecuado y lo entrega al consumidor, IBM MQ classes for JMS reanuda la búsqueda del siguiente mensaje adecuado a partir de su posición actual en la cola. IBM MQ classes for JMS continúa buscando en la cola de este modo hasta que alcanza el final de la cola, o hasta que haya transcurrido el intervalo de tiempo en milisegundos, de acuerdo con lo establecido por el valor de esta propiedad. En cada caso, IBM MQ classes for JMS vuelve al principio de la cola para continuar su búsqueda, y comienza un nuevo intervalo de tiempo.
securityExit [3]	Serie	• **null** • Nombre completo de una clase que implementa la interfaz MQSecurityExit de IBM MQ classes for Java	Identifica un programa de salida de seguridad de canal
securityExitInit	Serie	• **null** • Una serie de datos de usuario	Los datos de usuario que se pasan a un programa de salida de seguridad de canal, cuando se le invoca
sendExit [3]	Serie	• **null** • Serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa la interfaz MQSendExit de IBM MQ classes for Java	Identifica un programa de salida de canal de emisión o una secuencia de programas de salida de emisión que se deben ejecutar sucesivamente

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
sendExitInit	Serie	• **null** • Una serie que comprende uno o varios elementos de datos de usuario separados por comas	Datos de usuario que se pasan a programas de salida de envío de canal cuando se invocan los programas
shareConvAllowed	Boolean	• NO-Una conexión de cliente no puede compartir su socket. • **YES** -Una conexión de cliente puede compartir su socket.	Determina si una conexión de cliente puede compartir su socket con otras conexiones JMS de nivel superior del mismo proceso establecidas con el mismo gestor de colas, si las definiciones de canal coinciden
sparseSubscriptions [1]	Boolean	• **false** - Las suscripciones reciben mensajes coincidentes frecuentes. • true - Las suscripciones reciben mensajes coincidentes poco frecuentes. Este valor exige que la cola de suscripciones se pueda abrir para examinarla.	Controla la política de recuperación de mensajes de un objeto TopicSubscriber
sslCertStores	Serie	• **null** • Una serie de caracteres de uno o más URL de LDAP, separados por espacios en blanco. Cada URL de LDAP tiene el formato: `ldap://host_name [: port ]` donde *nombre_host* es el nombre o dirección IP del host, *puerto* es un número de puerto TCP y los corchetes indican un componente opcional.	Servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados para ser utilizadas en una conexión TLS.
sslCipherSuite	Serie	• **null** • El nombre de un CipherSuite	CipherSuite que se debe utilizar para una conexión TLS
sslFipsRequired [2]	Boolean	• **false** • true	Determina si una conexión TLS debe utilizar una CipherSuite que está soportada por el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS)

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
sslPeerName	Serie	• **null** • Una plantilla para nombres distinguidos	Para una conexión TLS, plantilla que se utiliza para comprobar el nombre distinguido contenido en el certificado digital presentado por el gestor de colas
sslResetCount	int	• **0** • Un entero en el rango 0-999 999 999	Número total de bytes enviados y recibidos por una conexión TLS antes de que se renegocien las claves secretas utilizadas por TLS
sslSocketFactory	Serie	Serie de caracteres que representa el nombre completo de una clase que proporciona una implementación de la interfaz javax.net.ssl.SSLSocketFactory. Opcionalmente incluye un argumento que se debe pasar al método constructor, encerrado entre paréntesis.	Las conexiones establecidas en el ámbito del objeto administrado utilizan sockets obtenidos de esta implementación de la interfaz SSLSocketFactory.
statusRefreshInterval [1]	int	• **60000** • Cualquier entero positivo	Intervalo, en milisegundos, entre las actualizaciones de la transacción de larga ejecución que detecta cuando un suscriptor pierde su conexión con el gestor de colas. Esta propiedad sólo es aplicable si **subscriptionStore** tiene el valor QUEUE.
subscriptionStore [1]	Serie	• **Intermediario** • MIGRATE • COLA	Determina el lugar donde IBM MQ classes for JMS almacena datos persistentes sobre suscripciones activas

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
transportType	Serie	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	Determina si una conexión con un gestor de colas utiliza la modalidad de cliente o la modalidad de enlaces. Si se especifica el valor `BINDINGS_THEN_CLIENT`, el adaptador de recursos primero intenta establecer una conexión en la modalidad de enlaces. Si este intento de conexión falla, el adaptador de recursos intenta entonces establecer una conexión en la modalidad de cliente. z/OS Si se ha configurado una especificación de activación que se ejecuta en un sistema WebSphere Application Server for z/OS para utilizar la modalidad de transporte `BINDINGS_THEN_CLIENT` y se ha interrumpido una conexión establecida previamente, cualquier intento de reconexión de la especificación de activación primero intenta utilizar la modalidad de transporte `BINDINGS` . Si el intento de conexión para la modalidad de transporte `BINDINGS` falla, la especificación de activación intenta luego establecer una conexión en modalidad de transporte `CLIENT`.
username	Serie	• **null** • El nombre de un usuario	El nombre de usuario predeterminado a utilizar cuando se crea una conexión con el gestor de colas

Tabla 64. Propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS (continuación)			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
wildcardFormat	Serie	• CHAR - Reconoce solamente comodines a nivel de carácter, tal como se utiliza en broker versión 1 • **TOPIC** - Reconoce comodines a nivel de tema solamente, tal como se utiliza en broker versión 2	La versión de sintaxis de comodín que debe utilizarse

Notas:

1. Esta propiedad se puede utilizar con la versión 70 de IBM MQ classes for JMS.
2. Para conocer información importante sobre la utilización de la propiedad sslFipsRequired, consulte “Limitaciones del adaptador de recursos de IBM MQ” en la página 453.
3. Para obtener información sobre cómo configurar el adaptador de recursos para que pueda localizar una salida, consulte “Configuración de IBM MQ classes for JMS para utilizar salidas de canal” en la página 289.
4. V 9.3.0 La propiedad dynamicallyBalanced no está soportada junto con el soporte de transacciones XA. Si dynamicallyBalanced es "true", el MDB debe estar configurado para inhabilitar las transacciones XA.

Propiedades utilizadas para crear un consumidor de conexión de JMS

Nota: Las propiedades **destination** y **destinationType** se deben definir de forma explícita. Todas las demás propiedades descritas en la Tabla 65 en la página 477 son opcionales.

Tabla 65. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión de JMS			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
destino	Serie	El nombre de un destino	El destino desde donde recibir mensajes. La propiedad **useJNDI** determina cómo se interpreta el valor de esta propiedad.
destinationLookup	Serie	• **null** • El nombre JNDI de un objeto Destination	Si se establece esta propiedad, ActivationSpec busca un objeto Destination de JMS que tenga el nombre JNDI especificado en el espacio de nombres JNDI del servidor de aplicaciones y luego utiliza las propiedades de ese objeto para crear un consumidor de conexión de JMS, en preferencia a las demás propiedades especificadas en ActivationSpec. Para obtener más información, consulte “Propiedades connectionFactoryLookup y destinationLookup de ActivationSpec” en la página 481.

Tabla 65. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión de JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
destinationType	Serie	• V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) • V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	El tipo de destino: una cola o un tema
maxMessages	int	• **1** • Un entero positivo	Número máximo de mensajes que se pueden asignar cada vez a una sesión con el servidor. Si ActivationSpec está entregando mensajes a un bean controlado por mensaje en una transacción XA, se utiliza el valor 1 sin importar el valor de esta propiedad.
maxPoolDepth	int	• **10** • Un entero positivo	El número máximo de sesiones de servidor en la agrupación de sesiones de servidor, utilizadas por el consumidor de conexión
messageSelector	Serie	• **null** • Una expresión de selector de mensajes SQL92	Una expresión de selector de mensajes que especifica los mensajes que se van a entregar
nonASFTimeout	int	• **0** • Un entero positivo	Un valor positivo indica que se utiliza la entrega no ASF. El valor de esta propiedad es el tiempo, en milisegundos, durante el cual una solicitud get espera los mensajes que pueden no haber llegado todavía (una llamada get con espera). El valor predeterminado, 0, indica que se utiliza la entrega ASF. Este parámetro es válido si: • La aplicación se ejecuta en WebSphere Application Server 7.0 o posterior. • La aplicación se ejecuta en WebSphere Liberty utilizando el nivel adecuado de la característica de cliente wmqJms. Para obtener más información, consulte “Liberty y el adaptador de recursos de IBM MQ” en la página 454.

Tabla 65. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión de JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
nonASFRollbackEnabled	Boolean	• **false** - El mensaje se consume incluso si el MDB falla • true - Un error en el MDB hace que el mensaje se vuelva a colocar en la cola.	Determina si la entrega del mensaje se realiza dentro de un punto de sincronización de IBM MQ si el MDB no es transaccional. Esta propiedad no se tiene en cuenta si el MDB es transaccional o si **nonASFTimeout** está establecido en `0`.
poolTimeout	int	• **300000** • Un entero positivo	Tiempo, en milisegundos, que se mantiene abierta una sesión con el servidor no utilizada en la agrupación de sesiones del servidor antes de que se cierre por inactividad.
readAheadAllowed	int	• **DESTINATION** - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de cola o tema. • DISABLED - No se permite la lectura hacia adelante. • ENABLED - Se permite la lectura hacia adelante. • QUEUE - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de cola. • TOPIC - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de tema.	Indica si la hebra de examen de especificación de activación puede utilizar la lectura anticipada para examinar varios mensajes del destino en un almacenamiento intermedio interno, antes de pasar a las sesiones del servidor para un consumo destructivo. **Nota:** La habilitación de la lectura anticipada puede dar como resultado un aumento de los mensajes JMSCC0108 , o una reducción del rendimiento, o ambos, si la velocidad de proceso de MDB no puede mantenerse al día con la velocidad de navegación de los mensajes del destino.
readAheadClosePolicy	int	• **ALL** - Todos los mensajes del almacenamiento intermedio interno de lectura anticipada se entregan al MDB antes de que se detenga. • CURRENT - Solo se completa la invocación actual del MDB, y luego se descartan los mensajes que haya en el almacenamiento intermedio interno de lectura anticipada.	Determina lo que ocurre a los mensajes del almacenamiento intermedio interno de lectura anticipada cuando el administrador detiene el MDB.

Tabla 65. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión de JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
receiveCCSID	int	• **0** - Utilizar `Charset.defaultCharset` de JVM • 1208 - UTF-8 • Un identificador de juego de caracteres codificados soportado	La propiedad de destino que define el CCSID de destino para la conversión de mensajes del gestor de colas. El valor se pasa por alto a menos que **`receiveConversion`** se establezca en `QMGR`.
receiveConversion	Serie	• **CLIENT_MSG** • QMGR	La propiedad de destino que determina si el gestor de colas va a realizar la conversión de datos.
sharedSubscription	Boolean	• **False** - El MDB no debe abrir la suscripción como suscripción compartida. • True - el MDB debe abrir la suscripción como una suscripción compartida (con las reglas que implica JMS 2.0, consulte la especificación JMS 2.0 en Java.net).	Determina cómo se controla un MDB desde una suscripción compartida. Para obtener más información sobre cómo utilizar esta propiedad, consulte la “Ejemplos de cómo definir la propiedad sharedSubscription” en la página 484.
startTimeout	int	• **10 000** • Un entero positivo	Periodo de tiempo, en milisegundos, dentro del cual se debe iniciar la entrega de un mensaje a un MDB después de que se haya planificado el trabajo para entregar el mensaje. Si se sobrepasa este periodo de tiempo, el mensaje se restituye a la cola.
subscriptionDurability	Serie	• **NonDurable** - Se utiliza una suscripción no duradera para entregar mensajes a un MDB suscrito al tema. • Durable - Se utiliza una suscripción duradera para entregar mensajes a un MDB suscrito al tema.	Indica si se utiliza una suscripción duradera o no duradera para entregar mensajes a un MDB suscrito al tema.
subscriptionName	Serie	• **"" (serie vacía)** • El nombre de una suscripción	El nombre de la suscripción duradera

Tabla 65. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión de JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
useJNDI	Boolean	• **false** - La propiedad denominada destination se interpreta como el nombre de una cola o tema de IBM MQ. • true-La propiedad denominada destination se interpreta como el nombre de uno de los objetos siguientes en el espacio de nombres JNDI del servidor de aplicaciones: – V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) – V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated Determina cómo se interpreta el valor de la propiedad denominada destination **Nota:** Esta propiedad está en desuso en IBM MQ 9.0. En su lugar se debe utilizar La propiedad destinationLookup.

Conflictos y dependencias de propiedades

Un objeto ActivationSpec puede tener propiedades en conflicto. Por ejemplo, puede especificar propiedades de TLS para una conexión en la modalidad de enlaces. En este caso, el comportamiento viene determinado por el tipo de transporte y el dominio de mensajería, que es punto a punto o de publicación/suscripción, según lo determinado por la propiedad **destinationType**. No se tienen en cuenta las propiedades que no sean aplicables al tipo de transporte o dominio de mensajería especificados.

Si define una propiedad que requiere que se definan otras propiedades, pero no define estas últimas, el objeto ActivationSpec emite una excepción InvalidPropertyException cuando se invoca su método validate() durante el despliegue de un MDB. La excepción se notifica al administrador del servidor de aplicaciones de una manera que depende del servidor de aplicaciones. Por ejemplo, si establece la propiedad subscriptionDurability en Durable, lo que indica que desea utilizar suscripciones duraderas, también debe definir la propiedad **subscriptionName**.

Si se define tanto la propiedad **ccdtURL** como la propiedad **channel**, se emite una excepción InvalidPropertyException. Sin embargo, si sólo define la propiedad **ccdtURL** , dejando la propiedad denominada **channel** con su valor predeterminado de SYSTEM.DEF.SVRCONN, no se genera ninguna excepción y la tabla de definición de canal de cliente identificada por la propiedad **ccdtURL** se utiliza para iniciar una conexión JMS .

Propiedades connectionFactoryLookup y destinationLookup de ActivationSpec

La especificación JMS 2.0 aporta dos nuevas propiedades de ActivationSpec. Puede proporcionar las propiedades connectionFactoryLookup y destinationLookup con un nombre de JNDI de un objeto administrado para que se utilicen en preferencia a otras propiedades de ActivationSpec.

Por ejemplo, si una fábrica de conexiones se define en JNDI y el nombre JNDI de ese objeto se especifica en la propiedad de búsqueda connectionFactorypara una especificación de activación, todas las propiedades de la fábrica de conexiones definidas en JNDI se utilizan de preferencia a las propiedades en Tabla 64 en la página 466.

Si un destino se define en JNDI y el nombre JNDI se establece en la propiedad destinationLookup de ActivationSpec, los valores de los que se utilizan en preferencia a los valores de Tabla 65 en la página 477. Para obtener más información sobre cómo se utilizan estas dos propiedades, consulte “Propiedades connectionFactoryLookup y destinationLookup de ActivationSpec” en la página 481.

Estas dos propiedades se pueden utilizar para especificar los nombres de JNDI de los objetos ConnectionFactory y Destination que se utilizan en preferencia a las propiedades de ActivationSpec tal como están definidas en la Tabla 64 en la página 466 y la Tabla 65 en la página 477.

Es importante tener en cuenta la información siguiente que describe con detalles cómo actúan estas propiedades.

connectionFactoryLookup
El valor de ConnectionFactory que se busca en JNDI se utiliza como origen de las propiedades listadas en Tabla 64 en la página 466. El objeto ConnectionFactory no se utiliza para crear realmente ninguna conexión de JMS, sólo se consultan las propiedades del objeto. Estas propiedades procedentes de ConnectionFactory prevalecen sobre las propiedades que están definidas en ActivationSpec. Hay una excepción a este comportamiento. Si ActivationSpec tiene establecida la propiedad **ClientID**, el valor de esta propiedad prevalece sobre el valor especificado en ConnectionFactory. Esto es así porque una situación habitual es la utilización de una sola ConnectionFactory con varias ActivationSpecs. Esto simplifica la administración. Pero la especificación JMS 2.0 establece que cada conexión JMS creada a partir de una fábrica de conexiones debe tener un **ClientID** exclusivo. Debido a esto, una ActivationSpec debe poder sustituir cualquier valor establecido en la fábrica de conexiones. Si no se establece ningún **ClientID** en ActivationSpec, se utiliza cualquier valor definido en la fábrica de conexiones.

destinationLookup
En la ActivationSpec se define una propiedad **Destination** y **UseJndi**. Si el distintivo **UseJndi** se establece en `true`, se considera que el texto especificado en la propiedad destination es un nombre de JNDI y se busca un objeto de destino con ese nombre de JNDI en el espacio de nombres de JNDI.

La propiedad destinationLookup se comporta exactamente de la misma manera. Si la propiedad se ha establecido, se busca un objeto de destino que tenga el nombre de JNDI especificado por esta propiedad en el espacio de nombres de JNDI. Esta propiedad tiene prioridad sobre la propiedad **useJNDI**.

Deprecated La propiedad useJNDI está en desuso en IBM MQ 9.0 ya que la propiedad **destinationLookup** es la especificación JMS 2.0 o posterior equivalente a realizar la misma función.

Propiedades de ActivationSpec sin equivalentes en IBM MQ classes for JMS

La mayoría de las propiedades de un objeto ActivationSpec son equivalentes a las propiedades de los objetos IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging , o a los parámetros de los métodos IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging . Sin embargo, tres propiedades de ajuste y una propiedad de usabilidad no tienen equivalentes en IBM MQ classes for JMS o IBM MQ classes for Jakarta Messaging:

startTimeout
El tiempo, en milisegundos, que el gestor de trabajos del servidor de aplicaciones espera a que los recursos estén disponibles, después de que el adaptador de recursos planifique un objeto Work para que entregue un mensaje a un MDB. Si este tiempo transcurre antes de que se inicie la entrega del mensaje, el objeto de trabajo excede el tiempo de espera, el mensaje se restituye a la cola y el adaptador de recursos puede intentar volver a entregar el mensaje. Se escribe un aviso en el rastreo de diagnóstico, si está habilitado, pero en lo demás no afecta al proceso de entrega de mensajes. Esta condición normalmente sólo se produce cuando el servidor de aplicaciones está experimentando un

carga de trabajo muy elevada. Si esta condición se da con frecuencia, puede aumentar el valor de esta propiedad para dar más tiempo al gestor de trabajos para planificar la entrega de mensajes.

maxPoolDepth
Número máximo de sesiones de la agrupación de sesiones de servidor utilizadas por un consumidor de conexión. Cuando se crea una sesión de servidor, la sesión inicia una conversación con un gestor de colas. El consumidor de conexión utiliza una sesión de servidor para entregar un mensaje a un MDB. Una agrupación de sesiones de mayor profundidad permite que se entreguen más mensajes simultáneamente en situaciones de mucha carga de trabajo. pero utiliza más recursos del servidor de aplicaciones. Si se deben desplegar muchos MDB, puede reducir la profundidad de la agrupación de sesiones para que la carga de trabajo en el servidor de aplicaciones se mantenga a un nivel manejable. Cada consumidor de conexión utiliza su propia agrupación de sesiones de servidor, por lo que esta propiedad no define el número total de sesiones de servidor disponibles para todos los consumidores de conexión.

poolTimeout
Tiempo, en milisegundos, que se mantiene abierta una sesión de servidor no utilizada en la agrupación de sesiones de servidor antes de que se cierre por inactividad. Un aumento transitorio de la carga de trabajo de mensajes hace que se creen sesiones de servidor adicionales para distribuir la carga, pero cuando la carga de trabajo vuelve a un nivel normal, las sesiones de servidor adicionales permanecen en la agrupación y no se utilizan.

Cada vez que se utiliza una sesión de servidor, se marca con una indicación de la fecha y hora. Periódicamente, una hebra limpiadora comprueba que cada sesión de servidor se haya utilizado dentro del periodo de tiempo especificado por esta propiedad. Si una sesión de servidor no se ha utilizado, se cierra y se elimina de la agrupación de sesiones de servidor. Es posible que una sesión de servidor no se cierre inmediatamente después de que haya transcurrido el periodo de tiempo especificado, esta propiedad representa el periodo mínimo de tiempo de inactividad antes de que se elimine.

useJNDI
Para obtener una descripción de esta propiedad, consulte la Tabla 65 en la página 477.

Despliegue de un MDB

Para desplegar un MDB, primero defina las propiedades de un objeto ActivationSpec, especificando las propiedades que necesita el MDB. El ejemplo siguiente es un conjunto típico de propiedades que puede definir de forma explícita:

V 9.3.0 JM 3.0 V 9.3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

El servidor de aplicaciones utiliza las propiedades para crear un objeto ActivationSpec que, posteriormente, se asocia a un MDB. Las propiedades del objeto ActivationSpec determinan cómo se entregan los mensajes al MDB. El despliegue del MDB falla si éste necesita transacciones distribuidas pero el adaptador de recursos no soporta las transacciones distribuidas. Para obtener información sobre

cómo instalar el adaptador de recursos de forma que se dé soporte a las transacciones distribuidas, consulte “Instalación del adaptador de recursos de IBM MQ” en la página 457.

Si más de un MDB recibe mensajes del mismo destino, un mensaje enviado en el dominio punto a punto sólo lo recibe un MDB, aunque haya otros MDB que pueden recibirlo. En concreto, si dos MDB utilizan selectores de mensajes distintos y un mensaje de entrada coincide con ambos selectores, sólo uno de los MDB recibe el mensaje. El MDB elegido para recibir el mensaje no está definido, por lo que no se puede predecir qué MDB lo recibirá. Los mensajes que se envían en el dominio de publicación/suscripción los reciben todos los MDB elegibles.

En algunas circunstancias, un mensaje entregado a un MDB se podría retrotraer en una cola IBM MQ. Por ejemplo, esto puede suceder si se entrega un mensaje dentro de una unidad de trabajo que posteriormente se retrotrae. Un mensaje que se restituye a la cola se entrega de nuevo, pero un mensaje mal formateado puede hacer que un MDB falle repetidamente y, por lo tanto, no se puede entregar. Dicho mensaje se denomina mensaje dañado. Puede configurar IBM MQ para que IBM MQ classes for JMS transfiera automáticamente un mensaje no entregable a otra cola para realizar una investigación adicional o descartar el mensaje.

Para obtener detalles sobre cómo manejar mensajes no entregables, consulte “Manejo de mensajes no entregables en IBM MQ classes for JMS” en la página 240.

Conceptos relacionados

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Tareas relacionadas

Configurar recursos de JMS en WebSphere Application Server

Ejemplos de cómo definir la propiedad sharedSubscription

Puede definir la propiedad sharedSubscription de una especificación de activación dentro de un archivo WebSphere Liberty `server.xml` . Como alternativa, también puede definir la propiedad dentro de un bean controlado por mensaje (MDB) utilizando anotaciones.

Ejemplo: definición dentro de un archivo Liberty `server.xml`

Dentro de un archivo WebSphere Liberty `server.xml` , defina una especificación de activación tal como se muestra en el ejemplo siguiente. En este ejemplo se crea una suscripción compartida duradera a un gestor de cola en localhost/port 1490.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

Ejemplo: definición dentro de un MDB

También puede definir la propiedad sharedSubscription dentro del MDB mediante anotaciones, según se muestra en el ejemplo siguiente:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

El ejemplo siguiente muestra una parte del código MDB que utiliza el método de anotaciones:

V 9.3.0 JM 3.0 V 9.3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
```

```
      @ActivationConfigProperty(
         propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
      @ActivationConfigProperty(
         propertyName = "sharedSubscription", propertyValue = "TRUE"),
      @ActivationConfigProperty(
         propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
      @ActivationConfigProperty(
         propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
      @ActivationConfigProperty(
         propertyName = "sharedSubscription", propertyValue = "TRUE"),
      @ActivationConfigProperty(
         propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

Conceptos relacionados

Suscriptores y suscripciones

Durabilidad de suscripción

“Suscripciones clonadas y compartidas” en la página 335

En IBM MQ 8.0 o posterior, hay dos métodos para otorgar a varios consumidores acceso a la misma suscripción. Estos dos métodos son mediante suscripciones clonadas o mediante suscripciones compartidas.

Configuración del adaptador de recursos para la comunicación de salida

Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

Ejemplo de utilización de la comunicación de salida

Cuando se utiliza comunicación de salida, una aplicación que se ejecuta en el servidor de aplicaciones inicia una conexión con un gestor de colas y, a continuación, envía mensajes a sus colas y recibe mensajes de sus colas, de forma asíncrona. Por ejemplo, el método de servlet siguiente, doGet(), utiliza comunicación de salida:

V 9.3.0 JM 3.0 V 9.3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
```

```
...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

Cuando el servlet recibe una solicitud GET de HTTP, recupera un objeto ConnectionFactory y un objeto Queue del espacio de nombres JNDI, y utiliza los objetos para enviar un mensaje a una cola de IBM MQ. A continuación, el servlet recibe el mensaje que ha enviado.

Recursos necesarios para la comunicación de salida

Para configurar la comunicación de salida, defina los recursos de Java EE Connector Architecture (JCA) en las categorías siguientes:

- Las propiedades de un objeto ConnectionFactory, que el servidor de aplicaciones utiliza para crear un objeto ConnectionFactory de JMS.
- Las propiedades de un objeto de destino administrado, que el servidor de aplicaciones utiliza para crear un objeto de cola de JMS o un objeto de tema de JMS.

La manera en que define estas propiedades depende de las interfaces de administración proporcionadas por el servidor de aplicaciones. Los objetos ConnectionFactory, Queue y Topic creados por el servidor de aplicaciones están enlazados en un espacio de nombres JNDI desde donde una aplicación puede recuperarlos.

Normalmente, define un objeto ConnectionFactory para cada gestor de colas al que las aplicaciones puedan necesitar conectarse. Define un objeto de cola para cada cola a las que las aplicaciones puedan necesitar acceder en el dominio punto a punto. Y define un objeto de tema para cada tema en el que las aplicaciones puedan desear publicar o suscribirse. Un objeto ConnectionFactory puede ser independiente del dominio. Como alternativa, puede ser específico del dominio, un objeto QueueConnectionFactory para el dominio punto a punto o un objeto TopicConnectionFactory para el dominio de publicación/suscripción.

Consejo: En JMS 2.0, se puede utilizar una fábrica de conexiones para crear tanto conexiones como contextos. Como resultado, es posible tener una agrupación de conexiones asociada a una fábrica de conexiones que contenga una mezcla de conexiones y contextos. Se recomienda que una fábrica de conexiones solo se utilice para crear conexiones o crear contextos. Esto garantiza que la agrupación de conexiones de dicha fábrica de conexiones solo contenga objetos de un tipo, lo que hace que la agrupación sea más eficiente.

Propiedades de un objeto ConnectionFactory

La Tabla 66 en la página 487 lista las propiedades de un objeto ConnectionFactory. El servidor de aplicaciones utiliza estas propiedades para crear un objeto ConnectionFactory de JMS.

Tabla 66. Propiedades de un objeto ConnectionFactory

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
applicationName	Serie	• Nombre de la clase que realiza la llamada, si está disponible, ajustada para que no sea más larga de 28 caracteres. Si no está disponible, se utiliza la serie `WebSphere MQ Client for Java`.	Nombre con el que se registra una aplicación en el gestor de colas. Este nombre de aplicación se muestra mediante el mandato **`DISPLAY CONN MQSC/PCF`** (donde el campo se denomina **`APPLTAG`**) o en la pantalla **Conexiones de aplicación** de IBM MQ Explorer (donde el campo se denomina **App name**).
Cola brokerCCSub	Serie	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • El nombre de una cola	El nombre de la cola a partir de la cual un consumidor de conexión recibe mensajes de suscripción no duraderos.
Cola brokerControl	Serie	• **SYSTEM.BROKER.CONTROL.QUEUE** • El nombre de una cola	El nombre de la cola de control de intermediario.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
Cola brokerPub	Serie	• **SYSTEM.BROKER.DEFAULT.STREAM** • El nombre de una cola	El nombre de la cola donde se envían los mensajes publicados (el valor de la corriente de datos).
Gestor de brokerQueue	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	Nombre del gestor de colas en el que se ejecuta el intermediario.
Cola brokerSub	Serie	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • El nombre de una cola	El nombre de la cola a partir de la cual un consumidor de mensajes no duradero recibe mensajes. Consulte la propiedad BROKERSUBQ para obtener más información.
brokerVersion	Serie	• **unspecified** - Cuando el intermediario ha migrado de la V6 a la V7, establezca esta propiedad de tal modo que las cabeceras RFH2 ya no se utilicen. Después de la migración, esta propiedad ya no es relevante. • **V1** - Para utilizar un intermediario de publicación/suscripción de IBM MQ. Este valor es el valor predeterminado si TRANSPORT se establece en BIND o CLIENT. • **V2** - Para utilizar un intermediario de IBM Integration Bus en modalidad nativa. Este valor es el valor predeterminado si TRANSPORT se establece en DIRECT o DIRECTHTTP.	La versión del intermediario que se está utilizando.
ccdtURL	Serie	• **null** • Un localizador universal de recursos (URL)	URL que identifica el nombre y la ubicación del archivo que contiene la tabla de definición de canal de cliente (CCDT) y especifica cómo se puede acceder al archivo.
CCSID	Serie	• **819** • Identificador de juego de caracteres codificados soportado por la máquina virtual Java (JVM)	Identificador de juego de caracteres codificados para una conexión.
canal	Serie	• **SYSTEM.DEF.SVRCONN** • El nombre de un canal MQI	El nombre del canal MQI que se debe utilizar.
cleanupInterval	int	• **3 600 000** • Un entero positivo	Intervalo, en milisegundos, entre ejecuciones en segundo plano del programa de utilidad de limpieza de publicación/suscripción.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
cleanupLevel	Serie	• **SAFE** • NINGUNO • STRONG • FORCE • NONDUR	Nivel de limpieza de un almacenamiento de suscripción basado en intermediario.
clientID	Serie	• **null** • Un identificador de cliente	El identificador de cliente para una conexión.
cloneSupport	Serie	• **DISABLED** - Sólo se puede ejecutar una instancia de un suscriptor de temas duradero simultáneamente. • ENABLED - Se pueden ejecutar simultáneamente dos o más instancias del mismo suscriptor de tema duradero, pero cada instancia se debe ejecutar en una máquina virtual Java (JVM) separada.	Determina si dos o más instancias del mismo suscriptor de tema duradero se pueden ejecutar de forma simultánea.
connectionNameList	Serie	• **localhost(1414)** • Serie compuesta por elementos separados por comas donde cada elemento tiene el formato: `HOSTNAME(PORT)` donde *NOMBRE_HOST* es un nombre DNS o una dirección IP.	Lista de nombres de conexión TCP/IP utilizados para las comunicaciones de salida. **connectionNameList** reemplaza las propiedades **hostname** y **port**. Esta propiedad se utiliza para volver a conectar con gestores de colas de varias instancias. **connectionNameList** es similar en forma a **localAddress**, pero no se debe confundir con él. **localAddress** especifica las características de las comunicaciones locales, mientras que **connectionNameList** especifica cómo acceder a un gestor de colas remoto.
failIfQuiesce	Boolean	• **true** • falso	Determina si las llamadas a determinados métodos fallan si el gestor de colas está en estado de desactivación temporal.
headerCompression	Serie	• **NONE** • SYSTEM - Se realiza compresión de cabecera de mensaje RLE.	Lista de las técnicas que se pueden utilizar para comprimir datos de cabecera en una conexión.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
hostName	Serie	• **localhost** • El nombre de un host • Una dirección IP	Nombre de host o dirección IP del sistema en el que reside el gestor de colas. Cuando está especificada, la propiedad **connectionNameList** reemplaza a las propiedades **hostname** y **port**.
localAddress	Serie	• **null** • Una serie con el formato: `[ host_name ][(low_port [, high_port ])]` donde *nombre_host* es el nombre de un host o una dirección IP, *puerto_inferior* y *puerto_superior* son números de puertos TCP, y los delimitadores indican un componente opcional	Si se trata de una conexión directa a un gestor de colas, esta propiedad especifica una o ambas características siguientes: • La interfaz de red local que se utilizará • El puerto local o un rango de puertos locales que se utilizará **localAddress** es similar en forma a **connectionNameList**, pero no se debe confundir con él. **localAddress** especifica las características de las comunicaciones locales, mientras que **connectionNameList** especifica cómo acceder a un gestor de colas remoto.
messageCompression	Serie	• **NONE** • Una lista de uno o varios de los valores siguientes separados por caracteres en blanco: RLE ZLIBFAST ZLIBHIGH	Lista de las técnicas que se pueden utilizar para comprimir datos de mensaje en una conexión.
messageSelection	Serie	• **CLIENT** • BROKER	Determina si la selección de mensajes es realizada por IBM MQ classes for JMS o por el intermediario. La selección de mensajes por el intermediario no está soportada cuando **brokerVersion** tiene el valor 1.
Contraseña	Serie	• **null** • Una contraseña	Contraseña predeterminada que se debe utilizar cuando se crea una conexión con el gestor de colas.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
pollingInterval	int	• **5000** • Cualquier entero positivo	Si cada escucha de mensajes en una sesión no tiene un mensaje adecuado en cola, este valor es el intervalo máximo, en milisegundos, que transcurre antes de que cada escucha de mensajes intente de nuevo obtener un mensaje de su cola. Si con frecuencia sucede esto de que no haya disponible ningún mensaje adecuado para ninguno de los escuchas de mensajes de una sesión, considere aumentar el valor de esta propiedad. Esta propiedad es aplicable sólo si **TRANSPORT** tiene el valor `BIND` o `CLIENT`.
port	int	• **1414** • Un número de puerto TCP	El puerto en el que el gestor de colas está a la escucha. Cuando está especificada, la propiedad **connectionNameList** reemplaza a las propiedades **hostname** y **port**.
providerVersion	serie	• **sin especificar** • Serie de caracteres en uno de los formatos siguientes – V.R.M.F – V.R.M – V.R – V donde V, R, M y F son valores enteros mayores o iguales que cero.	Versión, release, nivel de modificación y fixpack del gestor de colas al que se debe conectar la aplicación.
Intervalo pubAck	int	• **25** • Un entero positivo	Número de mensajes publicados por un publicador antes de que IBM MQ classes for JMS solicite un acuse de recibo al intermediario.
queueManager	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	Nombre del gestor de colas al que se va a conectar.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
receiveExit [3]	Serie	• **null** • Serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa la interfaz MQReceiveExit de IBM MQ classes for Java.	Identifica un programa de salida de recepción de canal, o una secuencia de programas de salida de recepción que se deben ejecutar en sucesión.
receiveExitInit	Serie	• **null** • Una serie que comprende uno o varios elementos de datos de usuario separados por comas	Datos de usuario que se pasan a los programas de salida de recepción de canal cuando se invocan.
rescanInterval	int	• **5000** • Cualquier entero positivo	Cuando un consumidor de mensajes en el dominio punto a punto utiliza un selector de mensajes para seleccionar los mensajes que desea recibir, IBM MQ classes for JMS busca en la cola de IBM MQ mensajes adecuados en la secuencia determinada por el atributo **MsgDeliverySequence** de la cola. Cuando IBM MQ classes for JMS encuentra un mensaje adecuado y lo entrega al consumidor, IBM MQ classes for JMS reanuda la búsqueda del siguiente mensaje adecuado a partir de su posición actual en la cola. IBM MQ classes for JMS continúa buscando en la cola de este modo hasta que alcanza el final de la cola, o hasta que haya transcurrido el intervalo de tiempo en milisegundos, de acuerdo con lo establecido por el valor de esta propiedad. En cada caso, IBM MQ classes for JMS vuelve al principio de la cola para continuar su búsqueda, y comienza un nuevo intervalo de tiempo.
securityExit [3]	Serie	• **null** • Nombre completo de una clase que implementa la interfaz MQSecurityExit de IBM MQ classes for Java	Identifica un programa de salida de seguridad de canal.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
securityExitInit	Serie	• **null** • Una serie de datos de usuario	Datos de usuario que se pasan a un programa de salida de seguridad de canal cuando este se invoca.
sendCheckCount	int	• **0** • Cualquier entero positivo	Número de llamadas de envío que se deben permitir entre las comprobaciones de errores de transferencia asíncrona, dentro de una misma sesión no transaccional de JMS
sendExit [3]	Serie	• **null** • Serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa la interfaz MQSendExit de IBM MQ classes for Java	Identifica un programa de salida de canal de emisión o una secuencia de programas de salida de emisión que se deben ejecutar sucesivamente.
sendExitInit	Serie	• **null** • Una serie que comprende uno o varios elementos de datos de usuario separados por comas	Datos de usuario que se pasan a programas de salida de canal de emisión cuando se invocan.
shareConvAllowed	Boolean	• NO-Una conexión de cliente no puede compartir su socket. • **YES** -Una conexión de cliente puede compartir su socket.	Determina si una conexión de cliente puede compartir su socket con otras conexiones de JMS de nivel superior desde el mismo proceso al mismo gestor de colas, si las definiciones de canal coinciden.
sparseSubscriptions	Boolean	• **false** - Las suscripciones reciben mensajes coincidentes frecuentes. • true - Las suscripciones reciben mensajes coincidentes poco frecuentes. Este valor exige que la cola de suscripciones se pueda abrir para examinarla.	Controla la política de recuperación de mensajes de un objeto TopicSubscriber.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
sslCertStores	Serie	• **null** • Una serie de caracteres de uno o más URL de LDAP, separados por espacios en blanco. Cada URL de LDAP tiene el formato: `ldap://host_name [: port ]` donde *nombre_host* es el nombre o dirección IP del host, *puerto* es un número de puerto TCP y los corchetes indican un componente opcional.	Servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados (CRL) para su uso en una conexión TLS.
sslCipherSuite	Serie	• **null** • El nombre de un CipherSuite	CipherSuite que se debe utilizar para una conexión TLS.
sslFipsRequired [2]	Boolean	• **false** • true	Determina si una conexión TLS debe utilizar una CipherSuite que está soportada por el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS).
sslPeerName	Serie	• **null** • Una plantilla para nombres distinguidos	Para una conexión TLS, plantilla que se utiliza para comprobar el nombre distinguido en el certificado digital proporcionado por el gestor de colas.
sslResetCount	int	• **0** • Un entero en el rango 0-999 999 999	Número total de bytes enviados y recibidos por una conexión TLS antes de que se renegocien las claves secretas utilizadas por TLS.
sslSocketFactory	Serie	Serie que representa el nombre completo de una clase que proporciona una implementación de la interfaz javax.net.ssl.SSLSocketFactory, incluido opcionalmente un argumento que se debe pasar al método constructor, encerrado entre paréntesis.	Las conexiones establecidas en el ámbito de los sockets de uso de objeto de destino administrados se obtienen de esta implementación de la interfaz SSLSocketFactory.
Intervalo statusRefresh	int	• **60000** • Cualquier entero positivo	Intervalo, en milisegundos, entre las actualizaciones de la transacción de larga ejecución que detecta cuando un suscriptor pierde su conexión con el gestor de colas. Esta propiedad sólo es aplicable si **SUBSTORE** tiene el valor QUEUE.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
subscriptionStore	Serie	• **Intermediario** • MIGRATE • COLA	Determina dónde IBM MQ classes for JMS almacena datos persistentes sobre suscripciones activas.
targetClientMatching	Boolean	• **true** • falso	Determina si un mensaje de respuesta, enviado a la cola identificada por el campo de cabecera JMSReplyTo de un mensaje entrante, tiene una cabecera MQRFH2 solo si el mensaje entrante tiene una cabecera MQRFH2. También puede configurar esta propiedad para una especificación de activación. Para obtener más información, consulte “Configuración de la propiedad targetClientMatching para una especificación de activación” en la página 505.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
temporaryModel	Serie	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • Cualquier serie	Nombre de la cola modelo a partir de la cual se crean colas temporales de JMS. Utilice `SYSTEM.DEFAULT.MODEL.QUEUE` si se cumplen las dos sentencias siguientes: • La aplicación utiliza una cola temporal que aceptará mensajes no persistentes. • Una sola aplicación cada vez creará una cola temporal en el gestor de colas al que apunta ConnectionFactory. Observe que `SYSTEM.DEFAULT.MODEL.QUEUE` sólo se puede abrir mediante una sola aplicación cada vez. Utilice `SYSTEM.JMS.TEMPQ.MODEL` en las situaciones siguientes: • Cuando la aplicación utiliza una cola temporal que aceptará mensajes persistentes. • Si varias aplicaciones se pueden conectar al gestor de colas al que apunta ConnectionFactory y esas aplicaciones necesitan crear colas temporales al mismo tiempo. Defina una nueva cola modelo con el atributo **DEFPSIST** establecido en YES, y el atributo **DEFSOPT** establecido en SHARED en la situación siguiente: • Cuando la aplicación utiliza una cola temporal que aceptará mensajes no persistentes, y varias aplicaciones se conectarán al gestor de colas al que apunta ConnectionFactory, y esas aplicaciones necesitan crear colas temporales al mismo tiempo. Cuando se cree la nueva cola modelo, establezca la propiedad **`temporaryModel`** en el nombre de la nueva cola modelo.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
tempQPrefix	Serie	• **"" (serie vacía)** • Prefijo que se puede utilizar para formar el nombre de una cola dinámica de IBM MQ. Las reglas para formar el prefijo son las mismas que las reglas para formar el contenido del campo **DynamicQName** en un descriptor de objeto IBM MQ , MQOD de estructura, pero el último carácter no en blanco debe ser un asterisco (*). Si el valor de la propiedad es la serie vacía, IBM MQ classes for JMS utiliza el valor AMQ.* al crear una cola dinámica.	Prefijo que se utiliza para formar el nombre de una cola dinámica de IBM MQ.
tempTopicPrefix	Serie	Cualquier serie no nula que conste solamente de caracteres válidos para una serie de tema de IBM MQ	Al crear temas temporales, JMS genera una serie de tema con el formato "TEMP/*TEMPTOPICPREFIX*/*unique_id* ", o si esta propiedad se deja con el valor predeterminado, simplemente "TEMP/*unique_id*". La especificación de un **TEMPTOPICPREFIX** no vacío permite definir colas de modelo específicas para crear las colas gestionadas para suscriptores a temas temporales creados bajo esta conexión.

Tabla 66. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
transportType	Serie	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	Determina si una conexión con un gestor de colas utiliza la modalidad de cliente o la modalidad de enlaces. Si se especifica el valor `BINDINGS_THEN_CLIENT`, el adaptador de recursos primero intenta establecer una conexión en la modalidad de enlaces. Si este intento de conexión falla, el adaptador de recursos intenta entonces establecer una conexión en la modalidad de cliente. z/OS Si se ha configurado una especificación de activación que se ejecuta en un sistema WebSphere Application Server for z/OS para utilizar la modalidad de transporte `BINDINGS_THEN_CLIENT` y se ha interrumpido una conexión establecida previamente, cualquier intento de reconexión de la especificación de activación primero intenta utilizar la modalidad de transporte `BINDINGS` . Si el intento de conexión para la modalidad de transporte `BINDINGS` falla, la especificación de activación intenta luego establecer una conexión en modalidad de transporte `CLIENT`.
username	Serie	• **null** • El nombre de un usuario	Nombre de usuario predeterminado que se debe utilizar cuando se crea una conexión con un gestor de colas.
wildcardFormat	int	• CHAR - Reconoce solamente comodines a nivel de carácter, tal como se utiliza en broker versión 1 • TOPIC - Reconoce solamente comodines a nivel de tema, tal como se utiliza en broker versión 2	Determina la versión de la sintaxis para comodines que se debe utilizar.

Notas:

1. Para conocer información importante sobre la utilización de la propiedad sslFipsRequired, consulte “Limitaciones del adaptador de recursos de IBM MQ” en la página 453.

2. Para obtener información sobre cómo configurar el adaptador de recursos para que pueda localizar una salida, consulte “Configuración de IBM MQ classes for JMS para utilizar salidas de canal” en la página 289.

El ejemplo siguiente muestra un conjunto típico de propiedades de un objeto ConnectionFactory:

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Propiedades de un objeto de destino administrado

El servidor de aplicaciones utiliza las propiedades de un objeto de destino administrado para crear un objeto de cola de JMS o un objeto de tema de JMS.

La Tabla 67 en la página 499 lista las propiedades que son comunes a un objeto de cola y un objeto de tema.

Tabla 67. Propiedades que son comunes a un objeto de cola y un objeto de tema

Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
CCSID	Serie	• **1208** • Identificador de juego de caracteres codificados soportado por la máquina virtual Java (JVM)	Identificador de juego de caracteres codificados para el destino.
codificación	Serie	• **NATIVE** • Una serie de tres caracteres: – El primer carácter especifica la representación de enteros binarios: - *N* indica codificación normal. - *R* indica codificación inversa. – El segundo carácter especifica la representación de enteros de decimal empaquetado: - *N* indica codificación normal. - *R* indica codificación inversa. – El tercer carácter especifica la representación de números de coma flotante: - *N* indica codificación IEEE estándar. - *R* indica codificación IEEE inversa. - *3* indica codificación de zSeries. NATIVE es equivalente a la serie NNN.	Representación de enteros binarios, enteros decimales empaquetados y números de coma flotante para el destino.

Tabla 67. Propiedades que son comunes a un objeto de cola y un objeto de tema (continuación)

Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
caducidad	Serie	• **APP** - El tiempo de caducidad de un mensaje está determinado por el productor de mensajes. • UNLIM - El mensaje no caduca nunca. • 0 - El mensaje no caduca nunca. • Entero positivo que representa el tiempo de caducidad de un mensaje en milisegundos.	Tiempo de caducidad de un mensaje enviado al destino.
failIfQuiesce	Serie	• **true** • falso	Determina si un intento de acceder al destino falla cuando el gestor de colas está en un estado de desactivación temporal.

Tabla 67. Propiedades que son comunes a un objeto de cola y un objeto de tema (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
messageBodyStyle	Serie	• **UNSPECIFIED** • JMS • MQ	Puede establecer la propiedad **messageBodyStyle** en colas y temas de JMS : UNSPECIFIED (valor predeterminado) • En el envío, IBM MQ classes for JMS genera e incluye una cabecera MQRFH2, dependiendo del valor de WMQ_TARGET_CLIENT. • En la recepción, IBM MQ classes for JMS establece las propiedades de mensajes de JMS de acuerdo con los valores contenidos en la cabecera MQRFH2 ,si existe. MQRFH2 no se presenta como parte del cuerpo del mensaje de JMS. JMS • En el envío, IBM MQ classes for JMS genera automaticamente una cabecera MQRFH2 y la incluye en el mensaje de IBM MQ. • En la recepción, IBM MQ classes for JMS establece las propiedades de mensajes de JMS de acuerdo con los valores contenidos en la cabecera MQRFH2 ,si existe. MQRFH2 no se presenta como parte del cuerpo del mensaje de JMS. MQ • En el envío, IBM MQ classes for JMS no genera una cabecera MQRFH2. • En la recepción, IBM MQ classes for JMS presenta la cabecera MQRFH2 como parte del cuerpo del mensaje de JMS.

Tabla 67. Propiedades que son comunes a un objeto de cola y un objeto de tema (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
persistence	Serie	• **APP** - La persistencia de un mensaje está determinada por el productor de mensajes. • QDEF - La persistencia de un mensaje está determinada por el atributo **DefPersistence** de la cola de IBM MQ. • PERS - El mensaje es persistente. • NON - El mensaje es no persistente • HIGH - La persistencia de un mensaje está determinada por el atributo **NonPersistentMessageClass** de la cola de IBM MQ de acuerdo con lo descrito en "Mensajes persistentes de JMS" en la página 260.	Persistencia de un mensaje enviado al destino.
priority	Serie	• **APP** - La prioridad de un mensaje está determinada por el productor de mensajes. • QDEF - La prioridad de un mensaje está determinada por el atributo **DefPriority** de la cola de IBM MQ. • Un entero dentro del rango 0 (prioridad más baja) a 9 (prioridad más alta).	Prioridad de un mensaje enviado al destino.
putAsyncAllowed	Serie	• QUEUE - Determina si se permiten las transferencias asíncronas mediante la consulta de la definición de cola. • TOPIC - Determina si se permiten las transferencias asíncronas mediante la consulta de la definición de tema. • DESTINATION - Determina si se permiten las transferencias asíncronas mediante la consulta de la definición de cola o de tema. • DISABLED - Las transferencias asíncronas no están permitidas. • ENABLED - Las transferencias asíncronas están permitidas.	Determina si los productores de mensajes pueden utilizar transferencias asíncronas para enviar mensajes a este destino.

Tabla 67. Propiedades que son comunes a un objeto de cola y un objeto de tema (continuación)			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
readAheadAllowed	int	• **DESTINATION** - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de cola o tema. • DISABLED - No se permite la lectura hacia adelante. • ENABLED - Se permite la lectura hacia adelante. • QUEUE - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de cola. • TOPIC - Determina si se permite la lectura hacia adelante, haciendo referencia a la definición de tema.	Determina si los consumidores de mensajes y navegadores de colas pueden utilizar la lectura anticipada para obtener mensajes no persistentes del destino y colocarlos en un almacenamiento intermedio interno antes de recibirlos.
receiveCCSID	int	• **0** - Utilizar `Charset.defaultCharset` de JVM • 1208 - UTF-8 • Un identificador de juego de caracteres codificados soportado	La propiedad de destino que define el CCSID de destino para la conversión de mensajes del gestor de colas. El valor se pasa por alto a menos que **`receiveConversion`** se establezca en `QMGR`.
receiveConversion	Serie	• **CLIENT_MSG** • QMGR	La propiedad de destino que determina si el gestor de colas va a realizar la conversión de datos.
targetClient	Serie	• **JMS** - El destino de un mensaje es una aplicación de JMS. • MQ - El destino de un mensaje es una aplicación no JMS IBM MQ.	Determina si el destino de un mensaje enviado al destino es una aplicación de JMS. Un mensaje cuyo destino es una aplicación de JMS contiene una cabecera MQRFH2.

La Tabla 68 en la página 503 lista las propiedades que son específicas de un objeto Queue.

Tabla 68. Propiedades que son específicas de un objeto Queue			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
baseQueueManagerName	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	Nombre del gestor de colas al que pertenece la cola de IBM MQ subyacente.
baseQueueName	Serie	• **"" (serie vacía)** • El nombre de una cola	Nombre de la cola de IBM MQ subyacente.

La Tabla 69 en la página 504 lista las propiedades que son específicas de un objeto Topic.

Tabla 69. Propiedades que son específicas de un objeto Topic

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
baseTopicName	Serie	• **"" (serie vacía)** • El nombre de un tema	Nombre del tema subyacente.
brokerCCDurSubQueue >	Serie	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • El nombre de una cola	Nombre de la cola desde la que un consumidor de conexión recibe mensajes de suscripción duradera.
brokerDurSubQueue	Serie	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • El nombre de una cola	Nombre de la cola desde la que un suscriptor de tema duradero recibe mensajes. Consulte la propiedad **BROKEDURRSUBQ** en la documentación de IBM MQ Explorer para obtener más información.
Cola brokerPub	Serie	• **No establecido** • El nombre de una cola	El nombre de la cola donde se envían los mensajes publicados (el valor de la corriente de datos). El valor de esta propiedad altera temporalmente el valor de la propiedad **brokerPubQueue** del objeto ConnectionFactory. Pero si no establece el valor de esta propiedad, en su lugar se utiliza el valor de la propiedad **brokerPubQueue** del objeto ConnectionFactory.
brokerPubQueueManager	Serie	• **"" (serie vacía)** • Un nombre de gestor de colas	Nombre del gestor de colas al que pertenece la cola a la que se envían los mensajes publicados en el tema.
brokerVersion	Serie	• **No establecido** • 1 • 2	La versión del intermediario que se está utilizando. El valor de esta propiedad altera temporalmente el valor de la propiedad **brokerVersion** del objeto ConnectionFactory. Pero si no establece el valor de esta propiedad, en su lugar se utiliza el valor de la propiedad **brokerVersion** del objeto ConnectionFactory.

El ejemplo siguiente muestra un conjunto de propiedades de un objeto Queue:

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

El ejemplo siguiente muestra un conjunto de propiedades de un objeto Topic:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

Tareas relacionadas

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Configurar recursos de JMS en WebSphere Application Server

Referencia relacionada

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Configuración de la propiedad targetClientMatching para una especificación de activación

Puede configurar la propiedad **targetClientMatching** para una especificación de activación, de manera que la cabecera MQRFH2 se incluya en los mensajes de respuesta cuando los mensajes de solicitud no contengan una cabecera MQRFH2. Esto implica que las propiedades de mensaje que defina una aplicación sobre un mensaje de respuesta se incluirán cuando se envíe el mensaje.

Acerca de esta tarea

Si una aplicación de bean controlado por mensaje (MDB) consume mensajes que no contienen una cabecera MQRFH2, a través de una especificación de activación del adaptador de recursos JCA de IBM MQ, y posteriormente envía mensajes de respuesta al destino JMS creado a partir del campo `JMSReplyTo` del mensaje de respuesta, los mensajes de respuesta deben incluir una cabecera MQRFH2, incluso aunque los mensajes de solicitud no lo hagan, de lo contrario las propiedades de mensaje que haya definido la aplicación sobre un mensaje de respuesta se perderán.

La propiedad **targetClientMatching** indica si un mensaje de respuesta, enviado a la cola que identifica el campo de cabecera JMSReplyTo de un mensaje entrante, solo tiene una cabecera MQRFH2 si el mensaje entrante tiene una cabecera MQRFH2. Puede configurar esta propiedad para una especificación de activación, tanto en WebSphere Application Server traditional como en WebSphere Liberty.

Si establece el valor de la propiedad **targetClientMatching** en `false`, se puede incluir una cabecera MQRFH2 en un mensaje de respuesta enviado a un destino JMS creado a partir de la cabecera JMSReplyTo de un mensaje de solicitud entrante que no contenga una MQRFH2. Esto se debe a que la propiedad **targetClient** del destino JMS está establecida en el valor `0`, que implica que los mensajes contienen una cabecera MQRFH2. La presencia de la cabecera MQRFH2 en el mensaje de salida permite el almacenamiento de propiedades de mensaje definidas por el usuario en el mensaje cuando se envía a la cola IBM MQ.

Si la propiedad **targetClientMatching** se establece en `true` y un mensaje de solicitud no incluye una cabecera MQRFH2, no se incluirá una cabecera MQRFH2 en el mensaje de respuesta.

Procedimiento

- En WebSphere Application Server traditional, utilice la consola de administración para definir la propiedad **targetClientMatching** como una propiedad personalizada en la especificación de activación de IBM MQ:
  a) En el panel de navegación, pulse **Recursos -> JMS ->Especificaciones de activación**.
  b) Seleccione el nombre de la especificación de activación que desea ver o modificar.
  c) Pulse **Propiedades personalizadas -> Nueva** y, después, especifique los detalles de la nueva propiedad personalizada.

  Establezca el nombre de la propiedad en `targetClientMatching`, el tipo en `java.lang.Boolean` y el valor en `false`.

- En WebSphere Liberty, especifique la propiedad **targetClientMatching** en la definición de una especificación de activación dentro de `server.xml`.

  Por ejemplo:

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

Conceptos relacionados

"Creación de destinos en una aplicación de JMS" en la página 227
En lugar de recuperar destinos como objetos administrados desde un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación de JMS puede utilizar una sesión para crear destinos de forma dinámica en tiempo de ejecución. ´ Una aplicación puede utilizar un identificador uniforme de recursos (URI) para identificar una cola o tema de IBM MQ y, opcionalmente, especificar una o más propiedades de un objeto Queue o Topic.

"Configuración del adaptador de recursos para la comunicación de salida" en la página 485
Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

Poner en pausa el bean controlado por mensaje de IBM MQ en WebSphere Liberty

La propiedad **maxSequentialDeliveryFailures** para una especificación de activación define el número máximo de anomalías de entrega de mensajes secuenciales a una instancia de bean controlado por mensajes (MDB) que el adaptador de recursos tolera antes de poner en pausa el MDB.

Antes de empezar

Debe tener en cuenta el conjunto de sucesos que pueden hacer que un MDB se detenga en WebSphere Liberty. El adaptador de recursos considera cualquiera de las siguientes situaciones como un error de entrega de mensajes:

- Se emite una excepción no comprobada desde el método **onMessage** del MDB.
- Se produce una `JMSException` en el proceso del adaptador de recursos, antes de entregar el mensaje al MDB.
- Se produce una `JMSException` en el proceso del adaptador de recursos, antes de entregar el mensaje al MDB.
- Se retrotrae la transacción XA, o transacción local, utilizada para consumir el mensaje.
- No hay ninguna hebra disponible en el servidor de aplicaciones para entregar el mensaje al MDB.

Acerca de esta tarea

El valor predeterminado de la propiedad **maxSequentialDeliveryFailures** es *-1*, lo que significa que el MDB nunca está detenido. Cualquier otro valor negativo se trata igual que *-1*. Un valor de:

- *0* significa que el MDB se detiene en el primer error
- *1* significa que el MDB se detiene en dos errores secuenciales
- *2* significa que el MDB se detiene en tres errores secuenciales, etc.

Puede configurar esta propiedad para una especificación de activación, solo en WebSphere Liberty y cuando el nivel de Liberty sea 18.0.0.4 o superior.

Atención: si establece el atributo en un valor no predeterminado en cualquier entorno de servidor de aplicaciones que no sea Liberty, se omitirá el valor y se grabará un mensaje de advertencia en el registro.

Además, es posible instalar el adaptador de recursos de IBM MQ en WebSphere Liberty como un adaptador de recursos genérico. Con esto, se inhabilitan todas las capacidades de integración de IBM MQ y WebSphere Application Server y se impide que el adaptador de

recursos pueda detectar que se está ejecutando en Liberty. Por lo tanto, no se admite establecer **maxSequentialDeliveryFailures** en un valor mayor o igual que *0* y se genera un mensaje de advertencia en el registro.

Procedimiento

- En WebSphere Liberty, especifique la propiedad **maxSequentialDeliveryFailures** en la definición de una especificación de activación dentro de `server.xml`.

  Por ejemplo:

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

Conceptos relacionados

"Configuración del adaptador de recursos para la comunicación de salida" en la página 485
Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

Verificación de la instalación del adaptador de recursos

El programa de prueba de verificación de la instalación (IVT) para el adaptador de recursos de IBM MQ se proporciona en forma de archivo EAR. Para utilizar el programa, debe desplegarlo y definir algunos objetos como recursos de JCA.

Acerca de esta tarea

El programa de prueba de verificación de la instalación (IVT) se proporciona como un archivo EAR (Enterprise Archive) denominado `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) o `wmq.jmsra.ivt.ear` (JMS 2.0). Este archivo se instala con IBM MQ classes for JMS en el mismo directorio que el archivo RAR del adaptador de recursos de IBM MQ , `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) o `wmq.jmsra.rar` (JMS 2.0). Para obtener información sobre dónde se instalan estos archivos, consulte "Instalación del adaptador de recursos de IBM MQ" en la página 457.

Debe desplegar el programa IVT en el servidor de aplicaciones. El programa IVT incluye un servlet y un MDB que verifica si se puede enviar un mensaje a una cola de IBM MQ y recibirlo de ella. Puede utilizar el programa IVT para verificar que el adaptador de recursos de IBM MQ se ha configurado correctamente para dar soporte a las transacciones distribuidas. Si está desplegando el adaptador de recursos de IBM MQ en un servidor de aplicaciones que no es de IBM, el servicio técnico de IBM le puede solicitar que compruebe si el IVT está funcionando para verificar que el servidor de aplicaciones está configurado correctamente.

Antes de ejecutar el programa IVT, debe definir un objeto ConnectionFactory, un objeto Queue y posiblemente un objeto Activation Specification como recursos JCA y asegurarse de que el servidor de aplicaciones crea objetos JMS a partir de estas definiciones y los enlaza en un espacio de nombres JNDI. Puede elegir las propiedades de los objetos para que coincidan con los valores de host y puerto de su propio QueueManager, pero el conjunto de propiedades siguiente es un simple ejemplo:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

El mecanismo utilizado para definir los objetos ConnectionFactory, Queue y Activation Specification varía en función del servidor de aplicaciones. Por ejemplo, para establecer estas propiedades en WebSphere Liberty, añada las entradas siguientes al archivo `server.xml` del servidor de aplicaciones:

```
JM 3.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

```
JMS 2.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

De forma predeterminada, el programa IVT espera que un objeto ConnectionFactory esté enlazado en el espacio de nombres JNDI con el nombre jms/ivt/IVTCF y que un objeto Queue esté enlazado con el nombre jms/ivt/IVTQueue. Puede utilizar nombres distintos, pero si lo hace, debe especificar los nombres de los objetos en la página inicial del programa IVT y modificar el archivo EAR según sea necesario.

Una vez que haya desplegado el programa IVT y que el servidor de aplicaciones haya creado los objetos JMS y los haya enlazado en el espacio de nombres JNDI, puede iniciar el programa IVT siguiendo estos pasos:

Procedimiento

1. Inicie el programa IVT entrando un URL en el navegador web con el formato siguiente:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   donde *host_servidor_aplicaciones* es la dirección IP o nombre de host del sistema donde se está ejecutando el servidor de aplicaciones, y *puerto* es el número del puerto TCP en el que el servidor de aplicaciones está a la escucha. He aquí un ejemplo:

   ```
   http://localhost:9080/WMQ_IVT/
   ```

A continuación se muestra un ejemplo de la página inicial que muestra el programa IVT.

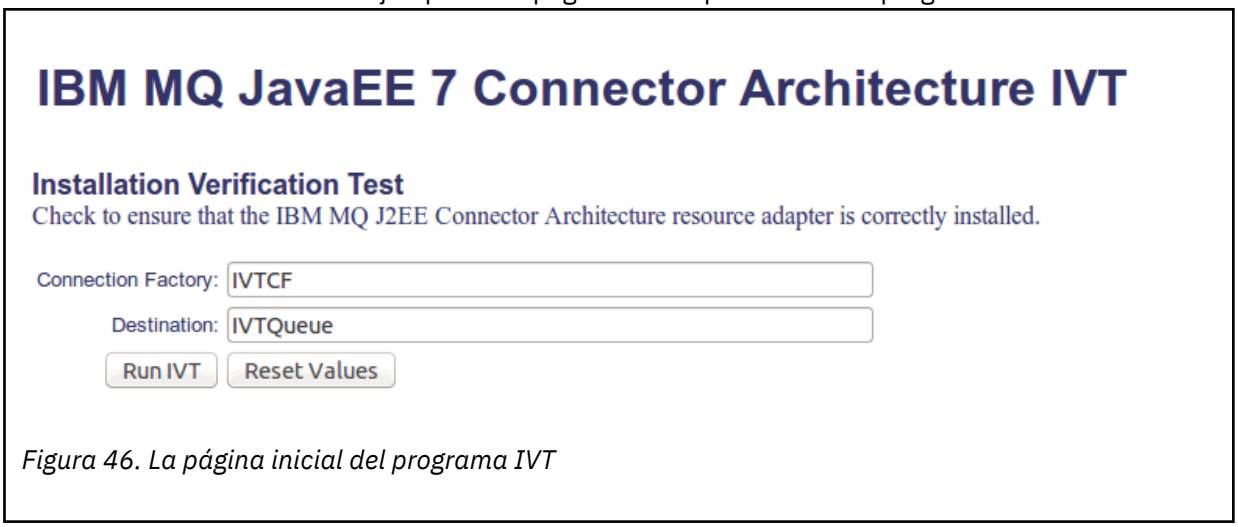


Figura 46. La página inicial del programa IVT

2. Para ejecutar la prueba, pulse **Ejecutar IVT**.

A continuación se muestra un ejemplo de la página que se muestra si la IVT es satisfactoria.

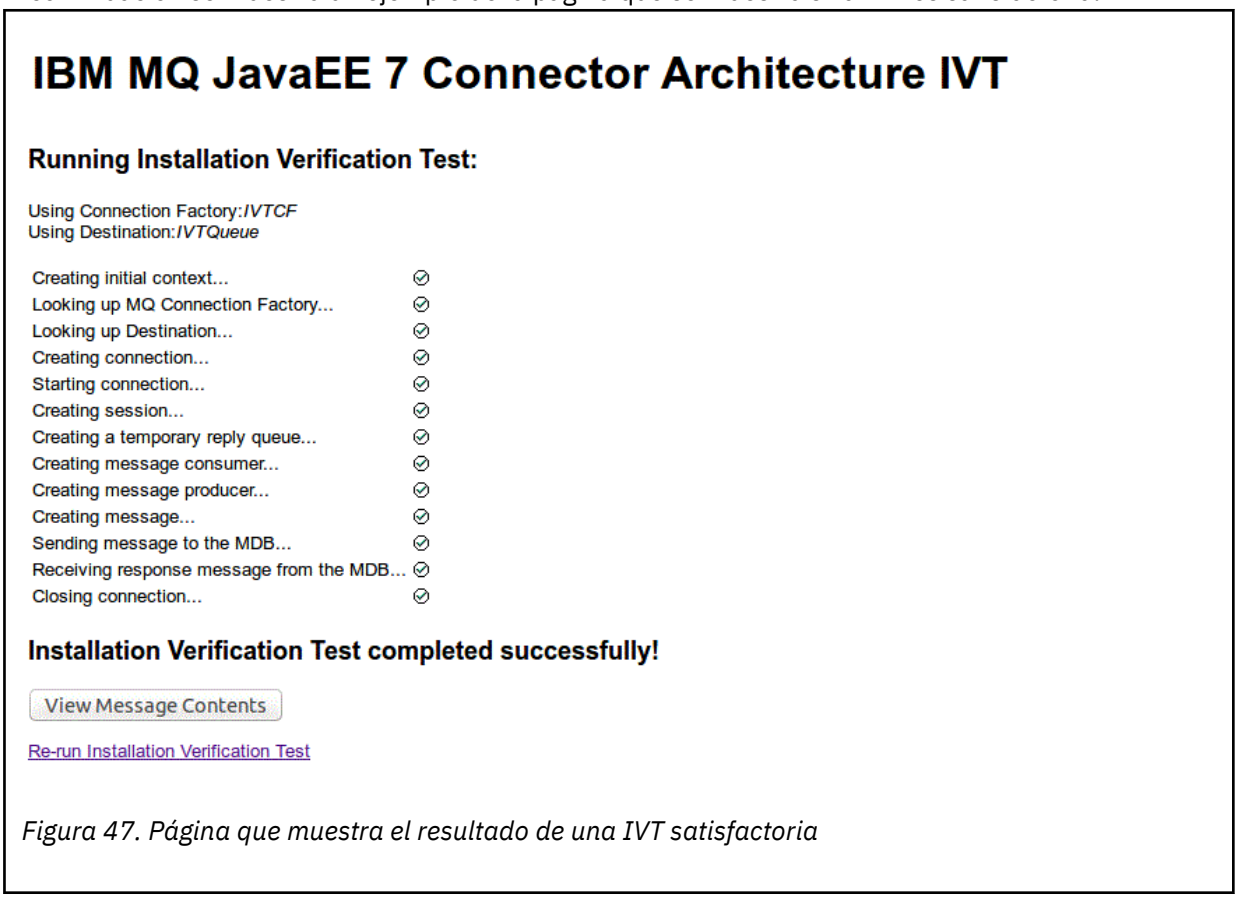


Figura 47. Página que muestra el resultado de una IVT satisfactoria

A continuación se muestra un ejemplo de la página que se muestra si la IVT falla. Para obtener información adicional sobre la causa del error, pulse **Ver rastreo de pila**.

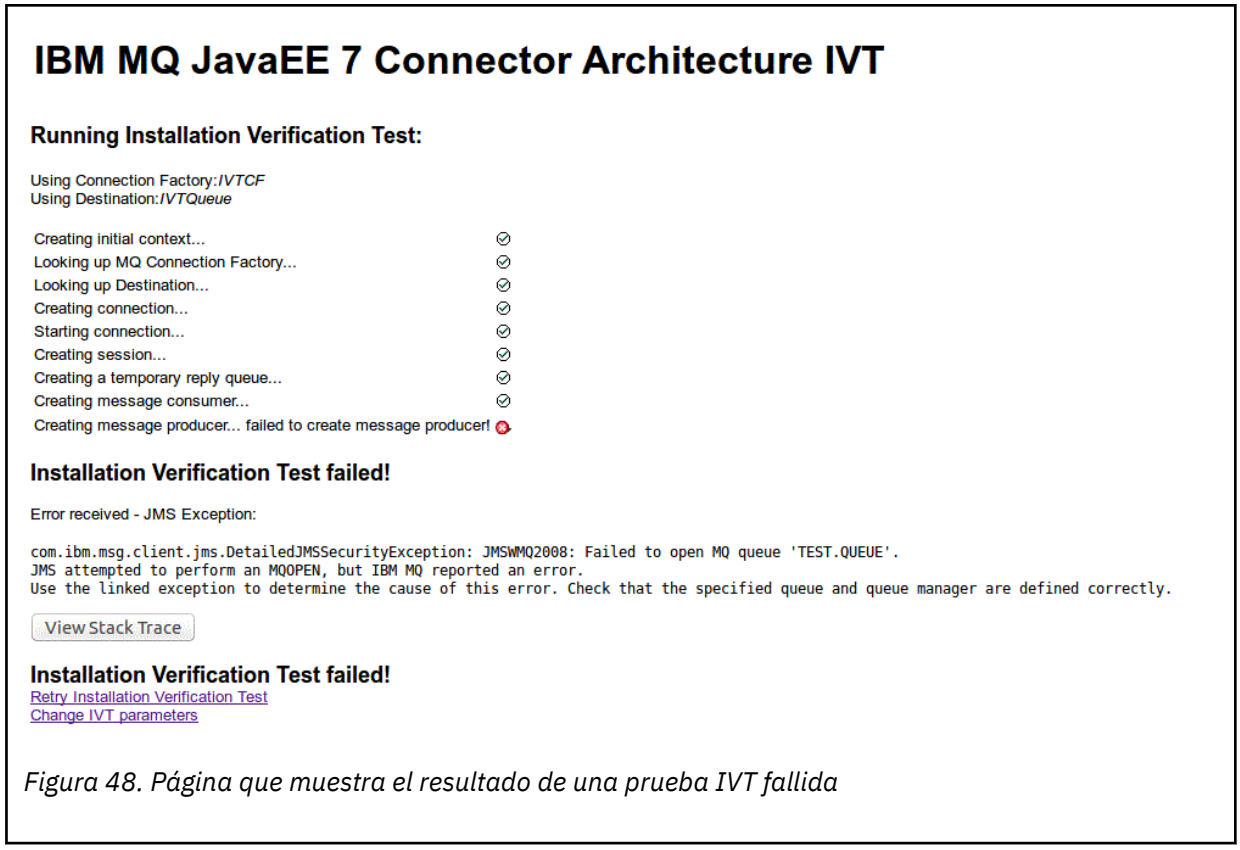


Figura 48. Página que muestra el resultado de una prueba IVT fallida

Windows Instalación y prueba del adaptador de recursos en GlassFish Server

Para instalar el adaptador de recursos IBM MQ en GlassFish Server en un sistema operativo Windows, en primer lugar, debe crear e iniciar un dominio. Luego se podrá desplegar y configurar el adaptador de recursos, y desplegar y ejecutar la aplicación de prueba de verificación de la instalación (IVT).

Antes de empezar

- Estas instrucciones corresponden a la versión 4 de GlassFish Server.
- Esta versión de GlassFish Server no da soporte a Jakarta EE.

Acerca de esta tarea

En esta tarea se asume que tiene un servidor de aplicaciones GlassFish Server en ejecución y que está familiarizado con las tareas de administración habituales del mismo. Esta tarea presupone que tiene una instalación de IBM MQ en el sistema local y que está familiarizado con las tareas de administración estándar.

Nota: Para poder completar los pasos de la tarea siguiente, debe tener una instalación de IBM MQ operativa con los objetos siguientes configurados:

- Un gestor de colas llamado QM, arrancado en el puerto 1414, que utilice el canal SYSTEM.DEF.SVRCONN y que se conecte mediante el transporte de cliente.
- Una cola llamada Q1.

Procedimiento

1. Inicie el programa de shell **asadmin** de GlassFish Server.
   a) Abra la línea de mandatos Windows y vaya hasta el directorio *GlassFish*/bin, donde *GlassFish* es el directorio donde se instala GlassFish Server versión 4.

b) Ejecute el comando **asadmin** en la línea de comandos.

El comando **asadmin** abre un programa de shell en la línea de comandos que permite crear un dominio.

GlassFish Server version 4 está arrancado en el sistema.

2. Cree e inicie un dominio.
   a) Utilice el comando **create-domain**, especificando el puerto y el nombre de dominio, para crear un dominio. Ejecute el siguiente comando en la línea de comandos:

```
create-domain --adminport port domain_name
```

donde *puerto* es el número de puerto y *nombre_dominio* es el nombre que quiera darle al dominio.

Nota: El comando **create-domain** tiene muchos parámetros opcionales asociados. Sin embargo, en esta tarea solo se necesita el parámetro `--adminport`. Para obtener más información, consulte la documentación del producto de GlassFish Server versión 4.

Si el puerto especificado está en uso, aparecerá el mensaje siguiente:

`El puerto de nombre_dominiopuerto está en uso`

Si el nombre de dominio especificado está en uso, recibirá un mensaje que indica que el nombre especificado ya está en uso, así como una lista de todos los nombres de dominio que no están disponibles en ese momento.

   b) Cuando se le solicite que especifique un nombre de usuario y una contraseña, especifique las credenciales que se vayan a usar para iniciar sesión en el servidor de aplicaciones a través de un navegador web.

Si el mandato se completa correctamente, se visualiza un mensaje que resume la creación del dominio en la línea de mandatos, incluido el mensaje `Command create-domain executed successfully.`

Ha creado un dominio correctamente.

   c) Inicie el dominio ejecutando el siguiente comando por línea de comandos:

```
start-domain domain_name
```

donde *nombre_dominio* es el nombre de dominio especificado anteriormente.

3. Utilice un navegador web para acceder al servidor de aplicaciones GlassFish.
   a) En la barra de direcciones de un navegador web, especifique lo siguiente:

```
localhost:port
```

donde *puerto* es el puerto especificado anteriormente al crear el dominio.

Se visualiza la consola de GlassFish.

   b) Cuando se haya cargado la consola de GlassFish y se le solicite un nombre de usuario y una contraseña, especifique las credenciales del paso 2b.

4. Suba el adaptador de recursos a GlassFish Server 4.
   a) En la barra de herramientas **Tareas comunes**, seleccione el elemento de menú**Aplicaciones** para que se visualice la página **Aplicaciones**.
   b) Pulse en el botón **Desplegar** para abrir la página **Desplegar aplicaciones o módulos**.
   c) Pulse el botón **Examinar** y, a continuación, vaya a la ubicación del archivo `wmq.jmsra.rar`. Seleccione el archivo y, a continuación, pulse **Aceptar**.

5. Cree una agrupación de conexiones.
   a) En la barra de herramientas, bajo **Recursos**, seleccione el elemento de menú **Conectores**.

b) Luego seleccione el elemento de menú **Agrupaciones de conexiones de conector** para abrir la página **Agrupaciones de conexiones de conector**.

c) Pulse **Nuevo** para abrir la página **Nueva agrupación de conexiones de conector (Paso 1 de 2)**.

d) En la página **Nueva agrupación de conexiones de conector (Paso 1 de 2)**, especifique el nombre jms/ivt/IVTCF-Connection-Pool en el campo **Nombre de agrupación**.

e) En el campo **Adaptador de recursos**, seleccione `wmq.jmsra`.

f) En el campo **Definición de conexión** , especifique `javax.jms.ConnectionFactory`.

g) Seleccione **Siguiente** y luego seleccione **Finalizar**.

6. Cree los recursos del conector.

a) En la barra de herramientas, en el menú **Conectores**, seleccione la opción **Recurso de conector** para abrir la página **Recursos de conector**.

b) Seleccione **Nuevo** para abrir la página **Nuevo recurso de conector**.

c) En el campo **Nombre JNDI**, especifique `IVTCF`.

d) En el campo **Nombre de agrupación**, especifique `jms/ivt/IVTCF-Connection-Pool`.

e) Deje vacíos los demás campos.

f) Por cada uno de los pares propiedad/valor siguientes, pulse **Añadir propiedad** y especifique el nombre de propiedad y el valor, tal como se muestra en el ejemplo siguiente:

- nombre: host; valor: localhost
- nombre: port; valor 1414
- nombre: channel; valor: SYSTEM.DEF.SVRCONN
- nombre: queueManager; valor: QM
- nombre: transportType; valor: CLIENT

Nota: Asegúrese de utilizar los valores correctos para sus valores de configuración, que pueden ser distintos de los que se muestran en este ejemplo.

g) En la barra de herramientas, en **Conectores**, seleccione el elemento de menú **Recursos de objeto de administrador** para abrir la página **Recursos de objeto de administrador**.

h) En la página **Recursos de objeto de administrador**, pulse **Nuevo** para abrir la página **Nuevo recurso de objeto de administrador**.

i) En el campo **Nombre JNDI**, especifique `IVTQueue`.

j) En el campo **Adaptador de recursos**, especifique `wmq.jmsra`.

k) En el campo **Tipo de recurso**, especifique `javax.jms.Queue`.

l) Deje el campo **Nombre de clase** tal como está.

m) Por cada uno de los pares propiedad/valor siguientes, pulse **Añadir propiedad** y especifique el nombre de propiedad y el valor, tal como se muestra en el ejemplo siguiente:

- name: name; valor: IVTQueue
- nombre: baseQueueManagerName; valor QM
- name: baseQueueName; valor: Q1

Nota: Asegúrese de utilizar los valores correctos para sus valores de configuración, que pueden ser distintos de los que se muestran en este ejemplo.

n) Pulse **Aceptar**.

o) Seleccione la casilla de verificación **Habilitado** y pulse **Habilitar**.

7. Despliegue el archivo EAR `wmq.jmsra.ivt.ear` en el servidor GlassFish.

a) Pulse en la opción **Aplicaciones** en la barra de herramientas para visualizar la página **Aplicaciones**.

b) Pulse **Desplegar** para añadir la aplicación IVT.

c) En el campo **Ubicación** , vaya a, y seleccione, `wmq.jmsra.ivt.ear`.

d) En el campo**Servidores virtuales**, seleccione **servidor** y pulse **Aceptar**.

8. Lance el programa IVT.
   a) Pulse en la opción **Aplicaciones** en la barra de herramientas para visualizar la página **Aplicaciones**.
   b) Pulse en `wmq.jmsra.ivt` en la tabla de aplicaciones desplegadas.
   c) Pulse en el botón **Lanzar** en la tabla Módulos y componentes.
   d) Seleccione el enlace http:.
   e) Pulse **Ejecutar IVT**.

   Ha iniciado el programa IVT y, si todo ha ido bien, aparecerá la salida siguiente:

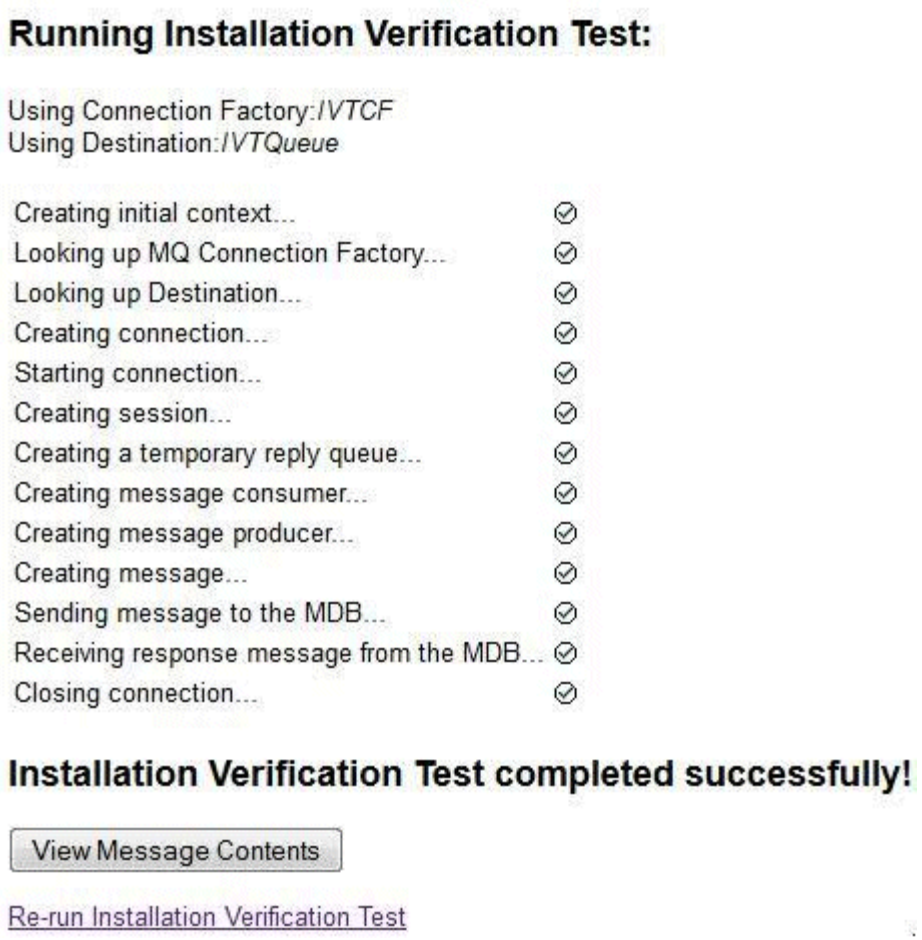


Figura 49. Salida de IVT satisfactoria

Instalar y probar el adaptador de recursos en Wildfly

Si está instalando el adaptador de recursos de IBM MQ en Wildfly V10, en primer lugar debe realizar algunos cambios en el archivo de configuración para añadir una definición del subsistema para el adaptador de recursos de IBM MQ. A continuación, puede desplegar el adaptador de recursos y probarlo instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Antes de empezar

- Estas instrucciones son para Wildfly V10.
- Esta versión de WildFly no da soporte a Jakarta EE.

Acerca de esta tarea

Esta tarea presupone que tiene un servidor de aplicaciones WildFly en ejecución y que está familiarizado con las tareas de administración estándar para el mismo. Esta tarea también presupone que tiene una instalación de IBM MQ y que está familiarizado con las tareas de administración estándar.

Procedimiento

1. Cree un gestor de colas de IBM MQ con el nombre ExampleQM, y configúrelo como se describe en la sección "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097.

   Cuando configure el gestor de colas, tenga en cuenta los puntos siguientes:

   - El escucha debe estar iniciado en el puerto 1414.
   - El canal que se ha de utilizar es SYSTEM.DEF.SVRCONN.
   - La cola que utiliza la aplicación IVT es TEST.QUEUE.

   También se debe conceder autorización de DSP y PUT a la cola modelo SYSTEM.DEFAULT.MODEL.QUEUE, de modo que esta aplicación pueda crear una cola de respuestas temporal.
2. Edite el archivo de configuración *WildFly_Home*/standalone/configuration/standalone-full.xml y añada el subsistema siguiente:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
```

```
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. Despliegue el adaptador de recursos en su servidor copiando el archivo `wmq.jmsra.rar` en el directorio *`WildFly_Home`*`/standalone/deployments`.
4. Despliegue la aplicación IVT copiando el archivo `wmq.jmsra.ivt.ear` en el directorio *`WildFly_Home`*`/standalone/deployments`.
5. Inicie el servidor de aplicaciones, activando un indicador de mandatos y ejecutando el mandato siguiente desde el directorio *`WildFly_Home`*`/bin`:

```
standalone.bat -c standalone-full.xml
```

6. Ejecute la aplicación IVT.

   Para obtener más información, consulte "Verificación de la instalación del adaptador de recursos" en la página 507. En Wildfly, el URL predeterminado es `http://localhost:8080/WMQ_IVT/`.

Utilización de IBM MQ y WebSphere Application Server juntos

A través del proveedor de mensajería de IBM MQ en WebSphere Application Server, las aplicaciones de mensajería de Java Message Service (JMS) pueden utilizar el sistema IBM MQ como proveedor externo de recursos de mensajería de JMS .

Acerca de esta tarea

Las aplicaciones que se escriben en Java y se ejecutan bajo WebSphere Application Server pueden utilizar la especificación Java Message Service (JMS) para realizar la mensajería. La mensajería en este entorno puede ser proporcionada por un gestor de colas de IBM MQ.

Una ventaja de utilizar un gestor de colas de IBM MQ es que las aplicaciones de JMS que se conectan pueden participar plenamente en la funcionalidad de una red de IBM MQ, lo que permite a las aplicaciones intercambiar mensajes con gestores de colas que se ejecutan en diversas plataformas.

Las aplicaciones pueden utilizar el *transporte de cliente* o el *transporte de enlaces* para el objeto de fábrica de conexiones de cola. Para el transporte de enlaces, el gestor de colas debe existir localmente en la aplicación que requiere una conexión.

De forma predeterminada, los mensajes de JMS que se almacenan en las colas de IBM MQ utilizan una cabecera MQRFH2 para contener parte de la información de cabecera de los mensajes de JMS. Muchas aplicaciones de IBM MQ heredadas no pueden procesar mensajes con estas cabeceras y requieren sus propias cabeceras características, por ejemplo, las aplicaciones MQCIH para CICS Bridge o MQWIH para IBM MQ Workflow. Para obtener más información sobre estas consideraciones especiales, consulte Correlación de mensajes de JMS en mensajes de IBM MQ.

Tareas relacionadas

Configurar recursos de JMS en WebSphere Application Server

Configurar el servidor de aplicaciones para que utilice el último nivel de mantenimiento del adaptador de recursos

Utilización de WebSphere Application Server con IBM MQ

IBM MQ y IBM MQ for z/OS se pueden utilizar con, o como una alternativa a, el proveedor de mensajería incluido con WebSphere Application Server.

El proveedor de mensajería de IBM MQ se instala como parte de WebSphere Application Server. Esto incluye una versión del adaptador de recursos de IBM MQ y la funcionalidad del cliente transaccional extendido de IBM MQ, que permite al gestor de colas participar en las transacciones XA gestionadas por el servidor de aplicaciones. Utilizando el adaptador de recursos, se pueden configurar los beans controlados por mensajes para que utilicen las especificaciones de activación o los puertos de escucha.

Para que se admita el servidor de aplicaciones, el programa de prueba de verificación de la instalación del adaptador de recursos de IBM MQ se debe desplegar en el servidor de aplicaciones y se debe ejecutar correctamente. Después de que el programa de prueba de verificación de la instalación del adaptador de recursos de IBM MQ se ha ejecutado correctamente, el adaptador de recursos de IBM MQ se puede conectar a cualquier gestor de colas de IBM MQ admitido.

Conexiones JMS de WebSphere Application Server a IBM MQ

Antes de considerar los niveles de IBM MQ que se pueden utilizar con WebSphere Application Server, es importante entender cómo las aplicaciones de Java Message Service (JMS) que se ejecutan dentro del servidor de aplicaciones se pueden conectar a los gestores de colas de IBM MQ.

Las aplicaciones JMS que deben acceder a los recursos de un gestor de colas IBM MQ pueden hacerlo utilizando uno de los tipos de transporte siguientes:

BINDINGS
Este transporte se puede utilizar cuando el servidor de aplicaciones y el gestor de colas están instalados en la misma máquina e imagen de sistema operativo. Cuando se utiliza la modalidad BINDINGS, toda la comunicación entre los dos productos se realiza mediante la comunicación entre procesos (IPC).

El proveedor de mensajería de IBM MQ no incluye las bibliotecas nativas necesarias para conectarse a un gestor de colas de IBM MQ en modalidad BINDINGS. Para poder utilizar una conexión de modalidad BINDINGS, IBM MQ debe estar instalado en la misma máquina que el servidor de aplicaciones y la vía de acceso de biblioteca nativa del adaptador de recursos se debe configurar para que apunte al directorio de IBM MQ donde están ubicadas estas bibliotecas. Para obtener más información, consulte la documentación del producto WebSphere Application Server:

- Para WebSphere Application Server traditional, consulte Configuración del proveedor de mensajería de IBM MQ con información de bibliotecas nativas.
- Para WebSphere Liberty, consulte Despliegue de aplicaciones JMS en Liberty para utilizar el proveedor de mensajería IBM MQ.

z/OS En z/OS, si desea conectar una fábrica de conexiones de WebSphere Application Server a un gestor de colas de IBM MQ en modalidad de enlaces, debe especificar las bibliotecas de IBM MQ correctas en la concatenación STEPLIB de WebSphere Application Server. Para obtener más información, consulte las bibliotecas de IBM MQ y WebSphere Application Server para z/OS STEPLIB en la documentación del producto WebSphere Application Server .

CLIENTE
El transporte de cliente utiliza TCP/IP para establecer comunicación entre WebSphere Application Server y IBM MQ. Además de utilizarse cuando el servidor de aplicaciones y el gestor de colas se encuentran en máquinas diferentes, la modalidad CLIENT también se puede utilizar cuando los dos productos están instalados en la misma máquina y en la misma imagen de sistema operativo.

Las aplicaciones de JMS también pueden especificar un tipo de transporte de BINDINGS_THEN_CLIENT. Cuando se utiliza este tipo de transporte, la aplicación intentará inicialmente conectarse al gestor de colas utilizando la modalidad BINDINGS, si no es capaz de hacerlo, se intentará el transporte CLIENT.

Cómo encontrar qué versión del adaptador de recursos de IBM MQ está instalada dentro de WebSphere Application Server

Para obtener información sobre qué versión del adaptador de recursos de IBM MQ está instalada dentro de WebSphere Application Server, consulte la nota técnica Which version of WebSphere MQ Resource

Adapter (RA) is shipped with WebSphere Application Server? (¿Qué versión del adaptador de recursos de WebSphere MQ se suministra con WebSphere Application Server?).

Puede utilizar los siguientes mandatos Jython y JACL para determinar el nivel del adaptador de recursos que WebSphere Application Server utiliza actualmente:

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

Nota: debe pulsar **Retorno** dos veces después de especificar este mandato para ejecutarlo.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Actualización del adaptador de recursos

Las actualizaciones del adaptador de recursos de IBM MQ que se instala con el servidor de aplicaciones se incluyen en los fixpacks de WebSphere Application Server. Actualización del adaptador de recursos de IBM MQ utilizando **Actualizar adaptador de recursos ...** recurso en la consola administrativa de WebSphere Application Server no se recomienda, ya que hacerlo significará que las actualizaciones proporcionadas en los fixpacks de WebSphere Application Server no tendrán ningún efecto.

Variable MQ_INSTALL_ROOT

A partir de WebSphere Application Server 7.0, MQ_INSTALL_ROOT solo se utiliza para localizar las bibliotecas nativas y se altera temporalmente mediante cualquier vía de acceso de biblioteca nativa configurada en el adaptador de recursos.

Conexión de WebSphere Application Server a IBM MQ

Atención:

1. Cualquier versión soportada de WebSphere Application Server puede utilizar el adaptador de recursos de IBM MQ que se entrega con él, para conectarse a cualquier versión soportada de IBM MQ.
2. Si se utiliza la modalidad de enlaces, determinadas bibliotecas en WebSphere Application Server deben coincidir con la versión del gestor de colas al que se conecta.
   - WebSphere Application Server se debe configurar para cargar las bibliotecas nativas proporcionadas con IBM MQ 9.3. Consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 100 para obtener más información.
   - z/OS En z/OS, debe especificar las bibliotecas IBM MQ correctas en la concatenación STEPLIB de WebSphere Application Server.

     Consulte Bibliotecas IBM MQ y WebSphere Application Server para z/OS STEPLIB si desea detalles de las bibliotecas IBM MQ que necesita.

     Si tiene bibliotecas para una versión de IBM MQ en LINKLIST (LINKLST), puede conectarse a una versión distinta de IBM MQ alterando temporalmente las bibliotecas con STEPLIB.
3. La versión del adaptador de recursos IBM MQ depende de las versiones de la biblioteca nativa (compartida) proporcionadas por la instalación del gestor de colas.

Por ejemplo, WebSphere Application Server 8.5, con un adaptador de recursos de IBM MQ 8.0 todavía puede gestionar una conexión de enlaces con un gestor de colas de IBM MQ 9.0 utilizando las bibliotecas nativas de IBM MQ 9.0 .

Para obtener más información, consulte “Sentencia de soporte del adaptador de recursos de IBM MQ” en la página 450.

Los tipos de transporte BINDINGS y CLIENT se pueden utilizar para conectarse a IBM MQ desde cualquier versión de WebSphere Application Server. Para el tipo de transporte BINDINGS, se aplican las restricciones siguientes:

- IBM MQ debe estar instalado en la misma máquina que el servidor de aplicaciones.
- WebSphere Application Server se debe configurar para cargar las bibliotecas nativas proporcionadas con IBM MQ.
- z/OS En z/OS, si desea conectar una fábrica de conexiones de WebSphere Application Server a un gestor de colas de IBM MQ en modalidad de enlaces, debe especificar las bibliotecas de IBM MQ correctas en la concatenación STEPLIB de WebSphere Application Server.

La tabla siguiente muestra las versiones de WebSphere Application Server en las que se admite la ejecución de cada versión del adaptador de recursos de IBM MQ .

Tabla 70. Correlación de versiones de WebSphere Application Server con versiones de adaptador de recursos de IBM MQ .

Versión de adaptador de recursos IBM MQ	**¿En qué versión de WebSphere Application Server se puede ejecutar esta versión del adaptador de recursos?**
IBM MQ 9.0 y posterior	El adaptador de recursos se puede ejecutar en: • Cualquier versión compatible con Java EE 7 de WebSphere Liberty. • WebSphere Application Server traditional 9.0
IBM MQ 8.0	El adaptador de recursos se puede ejecutar en cualquier versión compatible de Java EE 7 de WebSphere Liberty No se admite la ejecución del adaptador de recursos de IBM MQ 8.0 en WebSphere Application Server traditional. El adaptador de recursos ya instalado en WebSphere Application Server traditional se debe utilizar para conectarse a gestores de colas de IBM MQ 8.0.

Conceptos relacionados

“Sentencia de soporte del adaptador de recursos de IBM MQ” en la página 450
El adaptador de recursos IBM MQ que debe utilizar para la comunicación entre una aplicación y un gestor de colas depende de si está utilizando la API Jakarta Messaging 3.0 o la API JMS 2.0 .

Información relacionada

Requisitos de sistema para IBM MQ

Determinación del número de conexiones TCP/IP que se crean de WebSphere Application Server a IBM MQ

Al utilizar la funcionalidad de compartición de conversaciones, varias conversaciones pueden compartir instancias de canal MQI, que también se conoce como una conexión TCP/IP.

Acerca de esta tarea

Las aplicaciones que se ejecutan dentro de WebSphere Application Server 7 y WebSphere Application Server 8, que utilizan la modalidad normal de proveedor de mensajería de IBM MQ , utilizarán automáticamente esta característica. Esto significa que varias aplicaciones que se ejecutan en la misma instancia de servidor de aplicaciones, que se conectan al mismo gestor de colas IBM MQ, son capaces de compartir la misma instancia de canal.

El número de conversaciones que se pueden compartir entre una sola instancia de canal se determina mediante propiedad de canal IBM MQ **SHARECNV**. El valor predeterminado de esta propiedad para los canales de conexión de servidor es 10.

Al examinar el número de conversaciones creadas por WebSphere Application Server 7 y WebSphere Application Server 8, puede determinar el número de instancias de canal que se crean.

Para obtener más información sobre la modalidad del proveedor de mensajería IBM MQ, consulte Modalidad normal PROVIDERVERSION.

Conceptos relacionados

Utilización de las conversaciones compartidas
En un entorno en el que se permita compartir conversaciones, estas pueden compartir una instancia de canal MQI.

"Compartir una conexión TCP/IP en IBM MQ classes for JMS" en la página 325
Se pueden crear varias instancias de un canal MQI para que compartan una sola conexión TCP/IP.

Fábricas de conexiones de JMS

Las aplicaciones que se ejecutan dentro de WebSphere Application Server, que utilizan una fábrica de conexiones del proveedor de mensajería de IBM MQ para crear conexiones y sesiones, tienen conversaciones activas para cada conexión de JMS creada a partir de la fábrica de conexiones y para cada sesión de JMS creada a partir de una conexión de JMS .

Una conversación para cada conexión JMS creada desde la fábrica de conexiones

Cada fábrica de conexiones JMS tiene asociada una agrupación de conexiones, divida en dos secciones, la agrupación libre y la agrupación activa. Ambas agrupaciones están vacías inicialmente.

Cuando una aplicación crea una conexión JMS desde una fábrica de conexiones, WebSphere Application Server comprueba si hay una conexión JMS en la agrupación libre. Si la hay, ésta pasa a la agrupación activa y se otorga a la aplicación. De lo contrario, se crea una nueva conexión JMS, se coloca en la agrupación activa y se devuelve a la aplicación. El número máximo de conexiones que se pueden crear a partir de una fábrica de conexiones se especifica mediante la propiedad de agrupación de conexiones de fábrica de conexiones **Maximum connections**. El valor predeterminado de esta propiedad es 10.

Después de que una aplicación con una conexión JMS se haya finalizado y cerrado, la conexión pasa de la agrupación activa a la agrupación libre, donde está disponible para ser reutilizada. La propiedad **Unused timeout** de la agrupación de conexiones define durante cuánto tiempo puede estar una conexión JMS en la agrupación libre antes de su desconexión. El valor predeterminado de esta propiedad es 1800 segundos (30 minutos).

Cuando se crea por primera vez una conexión JMS se inicia una conversación entre WebSphere Application Server y IBM MQ. La conversación permanece activa hasta que se cierra la conexión cuando se supera el valor de la propiedad **Unused timeout** para la agrupación libre.

Una conversación para cada sesión JMS creada desde una conexión JMS

Cada conexión JMS creada desde una conexión de la fábrica de conexiones del proveedor de mensajería de IBM MQ tiene asociada una agrupación de sesiones JMS. Estas agrupaciones de sesiones funcionan del mismo modo que las agrupaciones de conexiones. El número máximo de sesiones de JMS que se pueden crear a partir de una única conexión de JMS viene determinado por la propiedad de agrupación de sesiones de fábrica de conexiones **Maximum connections**. El valor predeterminado de esta propiedad es 10.

Una conversación se inicia cuando se crea por primera vez una sesión de JMS . La conversación permanece activa hasta que se cierra la sesión de JMS porque ha permanecido en la agrupación libre durante más tiempo que el valor de la propiedad **Unused timeout** para la agrupación de sesiones.

Calcular un valor para la propiedad SHARECNV

Puede calcular el número máximo de conversaciones desde una única fábrica de conexiones con IBM MQ utilizando la fórmula siguiente:

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Puede utilizar la fórmula siguiente para calcular el número de instancias de canal que se crearán para permitir que se pueda llevar a cabo este número de conversaciones:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Se puede redondear cualquier resto de este cálculo.

Para una fábrica de conexiones simple que utiliza el valor predeterminado para las propiedades de agrupación de conexiones **Maximum connections** y agrupación de sesiones **Maximum connections** , el número máximo de conversaciones que pueden existir entre WebSphere Application Server y IBM MQ para esta fábrica de conexiones es:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Por ejemplo:

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Si está fábrica de conexiones se conecta con IBM MQ utilizando un canal que tiene establecida la propiedad **SHARECNV** en 10, el número máximo de instancias de canal que se creará para esta fábrica de conexiones es:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

Por ejemplo:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

Especificaciones de activación

Las aplicaciones de beans controlados por mensaje que se han configurado para que utilicen una especificación de activación tienen conversaciones activas con la especificación de activación, para supervisar un destino de JMS y para cada sesión de servidor que se utilice para ejecutar una instancia de bean controlado por mensaje que procesa los mensajes.

Las conversaciones siguientes están activas para las aplicaciones de beans controlados por mensajes que se han configurado para que utilicen una especificación de activación:

- Una conversación con la especificación de activación con el fin de supervisar un destino JMS para ver si hay mensajes adecuados. Esta conversación se inicia en cuanto se inicia la especificación de activación y permanece activa hasta que se detiene la especificación de activación.
- Una conversación para cada sesión de servidor utilizada para ejecutar un bean controlado por mensaje para procesar mensajes.

La propiedad avanzada de especificación de activación **Maximum server sessions** especifica el número máximo de sesiones de servidor que pueden estar activas en cualquier momento para una especificación de activación determinada. El valor predeterminado de esta propiedad es 10. Las sesiones de servidor se crean según sea necesario y se cierran si han estado desocupadas durante el periodo de tiempo que especifica la propiedad avanzada **Server session pool timeout** de la especificación de activación. El valor predeterminado de esta propiedad es de 300000 milisegundos (5 minutos).

Las conversaciones se inician cuando se crea una sesión de servidor y se detienen cuando se detiene la especificación de activación o cuando la sesión del servidor supera el tiempo de espera.

Esto significa que se puede calcular el número máximo de conversaciones desde una única especificación de activación para IBM MQ utilizando la fórmula siguiente:

```
Maximum number of conversations = Maximum server sessions + 1
```

Puede utilizar la fórmula siguiente para calcular el número de instancias de canal que se crearán para permitir que se pueda llevar a cabo este número de conversaciones:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Se puede redondear cualquier resto de este cálculo.

Para una especificación de activación simple, que utiliza el valor predeterminado para la propiedad **Maximum server sessions** , el número máximo de conversaciones que pueden existir entre WebSphere Application Server y IBM MQ para esta especificación de activación se calcula como:

```
Maximum number of conversations = Maximum server sessions + 1
```

Por ejemplo:

```
    = 10 + 1
    = 11
```

Si esta especificación de activación se conecta a IBM MQ utilizando un canal que tiene establecida la propiedad **SHARECNV** en 10, el número de instancias de canal que se crean se calcula de este modo:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Por ejemplo:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Puertos de escucha que se ejecutan en modalidad ASF (recursos del servidor de aplicaciones)

Los puertos de escucha que se ejecutan en modalidad ASF mediante aplicaciones de bean controlado por mensaje crean conversaciones para cada sesión de servidor. Uno supervisa un destino para ver si hay mensajes adecuados y otro ejecuta una instancia de bean controlado por mensaje para procesar

mensajes. El número de conversaciones para cada puerto de escucha se puede calcular a partir del número máximo de sesiones.

De forma predeterminada, los puertos de escucha se ejecutarán en modalidad ASF como parte de la especificación 1.1 que define el mecanismo que deben utilizar los servidores de aplicaciones para detectar mensajes y entregarlos a los beans controlados por mensaje para su proceso. Las aplicaciones de bean controlado por mensaje que estén configuradas para utilizar puertos de escucha en esta modalidad de operación predeterminada crean conversaciones:

Una conversación para que el puerto de escucha supervise un destino para comprobar si hay mensajes adecuados
: Los puertos de escucha están configurados para utilizar una fábrica de conexiones JMS. Cuando se inicia un puerto de escucha, se lleva a cabo una solicitud de una conexión JMS desde la agrupación libre de la fábrica de conexiones. Se devuelve la conexión a la agrupación libre cuando se detiene el puerto de escucha. Para obtener más información sobre cómo se utiliza la agrupación de conexiones y cómo afecta esto al número de conversaciones con IBM MQ, consulte "Fábricas de conexiones de JMS" en la página 519.

Una conversación para cada sesión de servidor utilizada para ejecutar una instancia de bean controlado por mensaje para procesar mensajes
: La propiedad **Maximum sessions** del puerto de escucha especifica el número máximo de sesiones de servidor que pueden estar activa en cualquier momento en un puerto de escucha concreto. El valor predeterminado de esta propiedad es 10. Las sesiones de servidor se crean a medida que se necesitan y hacen uso de sesiones JMS que se toman de la agrupación de sesiones asociada con la conexión JMS que está utilizando el puerto de escucha.

Si una sesión de servidor ha estado desocupada durante el periodo de tiempo especificado por la propiedad personalizada **SERVER.SESSION.POOL.UNUSED.TIMEOUT** del servicio de escucha de mensajes, la sesión se cierra y la sesión JMS utilizada se devuelve a la agrupación libre de la agrupación de sesiones. La sesión JMS permanecerá en la agrupación libre de la agrupación de sesiones hasta que sea necesaria, o se cerrará si ha estado desocupada en la agrupación libre durante más tiempo que el indicado por el valor de la propiedad **Unused timeout** de la agrupación de sesiones.

Para obtener más información sobre cómo se utiliza la agrupación de sesiones y cómo se gestionan las conversaciones entre WebSphere Application Server y IBM MQ, consulte "Fábricas de conexiones de JMS" en la página 519.

Para obtener más información sobre la propiedad personalizada **SERVER.SESSION.POOL.UNUSED.TIMEOUT** del servicio de escucha de mensajes, consulte Supervisión de agrupaciones de sesiones de servidor para puertos de escucha en la documentación del producto de WebSphere Application Server.

Cálculo del número máximo de conversaciones de un puerto de escucha individual con IBM MQ

Puede calcular el número máximo de conversaciones de un puerto de escucha individual con IBM MQ utilizando la fórmula siguiente:

```
Maximum number of conversations = Maximum sessions + 1
```

Puede utilizar la fórmula siguiente para calcular el número de instancias de canal que se crearán para permitir que se pueda llevar a cabo este número de conversaciones:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Se puede redondear cualquier resto de este cálculo.

Para un puerto de escucha simple que utiliza el valor predeterminado para la propiedad **Maximum sessions** , el número máximo de conversaciones que pueden existir entre WebSphere Application Server y IBM MQ para este puerto de escucha se calcula como:

```
Maximum number of conversations = Maximum sessions + 1
```

Por ejemplo:

```
    = 10 + 1
    = 11
```

Si este puerto de escucha se conecta a IBM MQ utilizando un canal que tiene la propiedad **SHARECNV** establecida en 10, el número de instancias de canal que se crearán se calcula como:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Por ejemplo:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Puertos de escucha que se ejecutan en modo no ASF (Application Server Facilities)

Los puertos de escucha que se ejecutan en modo no ASF se pueden configurar para que supervisen el destino de cola y el destino de tema utilizando las sesiones de servidor. Las sesiones de servidor pueden tener varias conversaciones, cuyo número máximo se puede calcular en cada caso.

Los puertos de escucha se pueden configurar para que se ejecuten en modo no ASF, lo cual cambia el modo en que los puertos de escucha supervisan los destinos JMS. Las aplicaciones de beans controlados por mensaje que utilizan puertos de escucha en modalidad de operación no ASF, crean una conversación para cada sesión de servidor que se utiliza para ejecutar una instancia de bean controlado por mensaje para procesar los mensajes. La propiedad **Maximum sessions** del puerto de escucha especifica el número máximo de sesiones de servidor que pueden estar activa en cualquier momento en un puerto de escucha concreto. El valor predeterminado para esta propiedad es 10.

Cuando un puerto de escucha que supervisa un destino de cola se ejecuta en modo no ASF, automáticamente crea el número de sesiones especificado en la propiedad **Maximum sessions** del puerto de escucha. Todas estas sesiones de servidor utilizan las sesiones JMS de la agrupación de sesiones asociada a la conexión JMS que utiliza el puerto de escucha y, de forma continuada, supervisan un destino JMS para obtener mensajes adecuados.

Si el puerto de escucha se ha configurado para que supervise un destino de tema, se omite el valor de **Maximum sessions** y se utiliza una sesión de servidor única.

Las sesiones de servidor que utiliza un puerto de escucha que se ejecuta en modo no ASF continúan activas hasta que se detiene el puerto de escucha y, a partir de ese momento, las sesiones JMS utilizadas se devuelven a la agrupación libre de sesiones para la conexión JMS que estaba utilizando el puerto de escucha.

Para obtener más información sobre cómo se utiliza la agrupación de sesiones y cómo se gestionan las conversaciones entre WebSphere Application Server y IBM MQ, consulte “Fábricas de conexiones de JMS” en la página 519.

Para obtener más información sobre la modalidad de operación ASF y no ASF con WebSphere Application Server, y sobre cómo configurar los puertos de escucha para utilizar la modalidad no ASF, consulte Proceso de mensajes en modalidad ASF y en modalidad no ASF.

Calcular el número máximo de conversaciones durante la supervisión de un destino de cola

El número máximo de conversaciones de un único puerto de escucha que se ejecuta en modo no ASF y supervisa una cola de destino para IBM MQ se puede calcular con la fórmula siguiente:

```
Maximum number of conversations = Maximum sessions
```

Puede utilizar la fórmula siguiente para calcular el número de instancias de canal que se crearán para permitir que se pueda llevar a cabo este número de conversaciones:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Se puede redondear cualquier resto de este cálculo.

En el caso de un puerto de escucha sencillo, en modo no ASF, que utiliza el valor predeterminado para la propiedad **Maximum sessions** y supervisa un destino de cola, el número máximo de conversaciones que pueden existir entre WebSphere Application Server e IBM MQ en este puerto de escucha es:

```
Maximum number of conversations = Maximum sessions
```

Por ejemplo:

```
     = 10
```

Si este puerto de escucha se conecta a IBM MQ utilizando un canal que tiene establecida la propiedad **SHARECNV** en 10, el número de instancias de canal que se crean se calcula de este modo:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Por ejemplo:

```
     = 10 / 10
     = 1
```

Calcular el número máximo de conversaciones durante la supervisión de un destino de tema

En el caso de un puerto que se ejecuta en modo no ASF y que se ha configurado para supervisar un destino, el número de conversaciones del puerto de escucha con IBM MQ es:

```
 Maximum number of conversations = 1
```

Puede utilizar la fórmula siguiente para calcular el número de instancias de canal que se crearán para permitir que se pueda llevar a cabo este número de conversaciones:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Se puede redondear cualquier resto de este cálculo.

En el caso de un puerto de escucha sencillo, en modo no ASF, que utiliza el valor predeterminado para la propiedad **Maximum sessions** y supervisa un destino de tema, el número máximo de conversaciones que pueden existir entre WebSphere Application Server e IBM MQ en este puerto de escucha es:

```
Maximum number of conversations = Maximum sessions
```

Por ejemplo:

```
     = 10
```

Si este puerto de escucha se conecta a IBM MQ utilizando un canal que tiene establecida la propiedad **SHARECNV** en 10, el número de instancias de canal que se crean se calcula de este modo:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Por ejemplo:

```
     = 10 / 10
     = 1
```

Configuración de alias para proteger la conexión de WebSphere Application Server a IBM MQ

Los alias de autenticación se correlacionan con una combinación de nombre de usuario y contraseña que se puede utilizar para proteger una conexión de WebSphere Application Server a IBM MQ. Puede configurar una fábrica de conexiones con un alias de autenticación.

Utilización de alias de autenticación con aplicaciones empresariales

Cuando una aplicación empresarial que se está ejecutando dentro de WebSphere Application Server intenta crear una conexión de JMS a IBM MQ, la aplicación busca una definición de fábrica de conexiones de proveedor de mensajería de IBM MQ en el repositorio Java Naming Directory Interface (JNDI) del servidor de aplicaciones.

Cuando la definición de fábrica de conexiones del proveedor de mensajería IBM MQ se encuentra dentro del repositorio JNDI del servidor de aplicaciones, se llama a uno de los métodos siguientes:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Si la fábrica de conexiones se ha configurado con un alias de autenticación J2C definido, el nombre de usuario y la contraseña del alias de autenticación puede fluir hasta IBM MQ cuando la fábrica de conexiones se utiliza para crear una conexión.

Fábricas de conexiones y alias de autenticación

Las fábricas de conexiones del proveedor de mensajería IBM MQ contienen información sobre cómo conectarse a gestores de colas IBM MQ. Las aplicaciones empresariales que se ejecutan dentro de WebSphere Application Server pueden utilizar las fábricas de conexiones para crear conexiones JMS a IBM MQ.

WebSphere Application Server almacena definiciones de fábricas de conexiones en un repositorio al que se puede acceder mediante JNDI. Cuando se crea una fábrica de conexiones, se proporciona un nombre JNDI a la fábrica de conexiones para identificarlo de forma exclusiva en el ámbito del servidor de aplicaciones (ya sea el ámbito de Célula, Nodo o Servidor), en el cual se ha definido.

Por ejemplo, una fábrica de conexiones de proveedor de mensajería IBM MQ definida en el ámbito de Célula de WebSphere Application Server contiene información sobre cómo conectarse al gestor de colas

(myQM) utilizando el transporte BINDINGS. Esta fábrica de conexiones recibe el nombre de JNDI jms/myCF para identificarla de forma exclusiva.

Las fábricas de conexiones también se pueden configurar para utilizar un alias de autenticación. Los alias de autenticación se correlacionan con una combinación de nombre de usuario y contraseña. En función de cómo se utiliza la fábrica de conexiones, el nombre de usuario y la contraseña del alias de autenticación podrían o no fluir hasta IBM MQ cuando se crea la conexión de JMS.

Importante: Antes de IBM MQ 8.0, el IBM MQ Object Authority Manager (OAM) predeterminado ha realizado una comprobación de autorización, solo para asegurarse de que el nombre de usuario pasado a IBM MQ, cuando se realiza una conexión, tenía la autorización para acceder al gestor de colas.

No se ha realizado ninguna comprobación para validar la contraseña que se ha especificado. Para poder realizar una comprobación de autenticación, y validar que el identificador de usuario y la contraseña coinciden, necesita escribir una salida de seguridad de canal de IBM MQ. Los detalles sobre cómo hacerlo se pueden encontrar en “Programas de salida de seguridad de canal” en la página 994.

A partir de IBM MQ 8.0, el gestor de colas comprueba la contraseña además del nombre de usuario.

Utilización de la fábrica de conexiones

Los temas siguientes contienen información sobre cómo utilizar la fábrica de conexiones utilizando búsquedas directas e indirectas:

- “Utilización de la fábrica de conexiones a través de una búsqueda directa” en la página 529
- “Utilización de la fábrica de conexiones a través de una búsqueda indirecta” en la página 530

Utilización del transporte CLIENT

Las fábricas de conexiones que se han configurado para utilizar el transporte CLIENT deben especificar qué canal de conexión de servidor IBM MQ (SVRCONN) van a utilizar para conectarse al gestor de colas.

Si la propiedad (MCAUSER) del identificador de usuario de agente de canal IBM MQ permanece en blanco para el canal que ha configurado la fábrica de conexiones para utilizarlo, la fábrica de conexiones se puede utilizar con una búsqueda directa o indirecta.

Si la propiedad MCAUSER está establecida en un identificador de usuario, este identificador de usuario se pasa hasta IBM MQ cuando se utiliza la fábrica de conexiones para crear una conexión a IBM MQ, independientemente de si la aplicación empresarial está utilizando una búsqueda directa o indirecta.

Tablas de resumen

Las tablas siguientes resumen qué identificadores de usuario se pasan a IBM MQ cuando se utilizan el transporte BINDINGS y el transporte CLIENT, respectivamente:

Tabla 71. Modalidad BINDINGS

Configuración	**Llamadas de aplicación `ConnectionFactory.createConnection()`**	**Llamadas de aplicación `ConnectionFactory.createConnection(String username, String password)`**
El descriptor de despliegue de la aplicación no contiene una referencia de recursos para la fábrica de conexiones.	El identificador de usuario para el proceso del servidor de aplicaciones fluye hasta IBM MQ.	El identificador de usuario y la contraseña que se han pasado en el método `ConnectionFactory.createConnection(String username, String password)` fluyen hasta IBM MQ.

Tabla 71. Modalidad BINDINGS (continuación)

Configuración	Llamadas de aplicación `ConnectionFactory.createConnection()`	Llamadas de aplicación `ConnectionFactory.createConnection(String username, String password)`
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones y la propiedad **res-auth** se establece en "Aplicación"	El identificador de usuario para el proceso del servidor de aplicaciones fluye hasta IBM MQ.	El identificador de usuario y la contraseña que se han pasado en el método `ConnectionFactory.createConnection(String username, String password)` fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones y la propiedad **res-auth** se establece en "Contenedor"	El identificador de usuario y la contraseña especificados en el alias de autenticación para la fábrica de conexiones fluyen hasta IBM MQ.	El identificador de usuario y la contraseña especificados en el alias de autenticación para la fábrica de conexiones fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **res-auth** establecida en "Contenedor" y la aplicación se ha configurado con un alias de autenticación	El identificador de usuario y la contraseña especificados en el alias de autenticación que ha configurado la aplicación para utilizarlo fluyen hasta IBM MQ.	El identificador de usuario y la contraseña especificados en el alias de autenticación que ha configurado la aplicación para utilizarlo fluyen hasta IBM MQ.

Tabla 72. Modalidad CLIENT

Configuración	Llamadas de aplicación `ConnectionFactory.createConnection()`	Llamadas de aplicación `ConnectionFactory.createConnection(String username, String password)`
El descriptor de despliegue de la aplicación no contiene una referencia de recursos para la fábrica de conexiones y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER sin establecer	El identificador de usuario para el proceso del servidor de aplicaciones fluye hasta IBM MQ.	El identificador de usuario y la contraseña que se han pasado en el método `ConnectionFactory.createConnection(String username, String password)` fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación no contiene una referencia de recursos para la fábrica de conexiones y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER establecida en un identificador de usuario	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.

Tabla 72. Modalidad CLIENT (continuación)		
Configuración	**Llamadas de aplicación `ConnectionFactory.createConnection()`**	**Llamadas de aplicación `ConnectionFactory.createConnection(String username, String password)`**
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en *Aplicación* y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER sin establecer	El identificador de usuario para el proceso del servidor de aplicaciones fluye hasta IBM MQ.	El identificador de usuario y la contraseña que se han pasado en el método `ConnectionFactory.createConnection(String username, String password)` fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en *Aplicación* y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER establecida en un identificador de usuario	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en "*Contenedor* y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER sin establecer	El identificador de usuario y la contraseña especificados en el alias de autenticación para la fábrica de conexiones fluyen hasta IBM MQ.	El identificador de usuario y la contraseña especificados en el alias de autenticación para la fábrica de conexiones fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en "*Contenedor* y la fábrica de conexiones está configurada para utilizar un canal IBM MQ que tiene la propiedad MCAUSER establecida en un identificador de usuario	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.

Tabla 72. Modalidad CLIENT (continuación)		
Configuración	**Llamadas de aplicación `ConnectionFactory.createConnection()`**	**Llamadas de aplicación `ConnectionFactory.createConnection(String username, String password)`**
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en "*Contenedor* y la aplicación se ha configurado con un alias de autenticación y la fábrica de conexiones se ha configurado para utilizar un canal IBM MQ que tiene la propiedad MCAUSER sin establecer	El identificador de usuario y la contraseña especificados en el alias de autenticación que ha configurado la aplicación para utilizarlo fluyen hasta IBM MQ.	El identificador de usuario y la contraseña especificados en el alias de autenticación que ha configurado la aplicación para utilizarlo fluyen hasta IBM MQ.
El descriptor de despliegue de la aplicación contiene una referencia de recursos para la fábrica de conexiones que tiene la propiedad **`res-auth`** establecida en *Contenedor* y la aplicación se ha configurado con un alias de autenticación y la fábrica de conexiones se ha configurado para utilizar un canal IBM MQ que tiene el MCAUSER establecido en un identificador de usuario	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.	El identificador de usuario especificado por la propiedad MCAUSER en el canal IBM MQ que se ha configurado para que lo utilice la fábrica de conexiones fluye hasta IBM MQ.

Utilización de la fábrica de conexiones a través de una búsqueda directa

Después de que se haya definido una fábrica de conexiones de proveedor de mensajería IBM MQ, una aplicación empresarial puede buscar la definición de fábrica de conexiones y utilizarla para crear una conexión JMS a un gestor de colas IBM MQ. Esto se realiza a través de una búsqueda directa.

Para utilizar la búsqueda directa, una aplicación empresarial se conecta al repositorio JNDI del servidor de aplicaciones, realizando la llamada de método siguiente:

```
InitialContext ctx = new InitialContext();
```

Una vez que se haya conectado al repositorio JNDI, la aplicación empresarial identifica la definición de fábrica de conexiones utilizando el nombre JNDI de la fábrica de conexiones, tal como se indica a continuación:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

Notas:

- El desarrollador de aplicaciones debe saber el nombre JNDI de la fábrica de conexiones necesaria cuando se está desarrollando la aplicación empresarial. Puesto que el nombre JNDI se ha codificado dentro de la aplicación, si el nombre JNDI cambia, tendrá que volverlo a escribir y volver a desplegar la aplicación.

- Cuando una definición de fábrica de conexiones se utiliza de esta forma, el nombre de usuario y la contraseña especificados en el alias de autenticación (que la fábrica de conexiones se ha configurado para utilizar) no fluyen hasta IBM MQ. Esto es para evitar que las aplicaciones no autorizadas identifiquen la fábrica de conexiones y puedan utilizarla para conectarse a sistemas IBM MQ seguros.

El nombre de usuario y la contraseña que fluyen hasta IBM MQ dependen del método que se utiliza para crear la conexión JMS de la fábrica de conexiones.

Si una aplicación crea una conexión JMS utilizando el método:

```
ConnectionFactory.createConnection()
```

la identidad del usuario predeterminado se pasa a IBM MQ. Este es la combinación de nombre de usuario y contraseña que ha iniciado el servidor de aplicaciones donde se está ejecutando la aplicación empresarial.

De forma alternativa, una aplicación puede crear una conexión JMS llamando al método:

```
ConnectionFactory.createConnection(String username, String password)
```

Si una aplicación ha realizado una búsqueda directa de una fábrica de conexiones y, después, ha llamado a este método. el nombre de usuario y la contraseña que se han pasado en el método `createConnection()` fluyen hasta IBM MQ.

Importante: Antes de IBM MQ 8.0, IBM MQ procesaba una comprobación de autorización, solo para asegurarse de que el nombre de usuario que se había pasado, tenía la autorización para acceder al gestor de colas.

No se realizaban comprobaciones respecto a la contraseña. Para poder realizar una comprobación de autenticación, y validar que el nombre de usuario y la contraseña eran válidos, se debe escribir una salida de seguridad de canal IBM MQ. Los detalles sobre cómo hacerlo se pueden encontrar en “Programas de salida de seguridad de canal” en la página 994.

A partir de IBM MQ 8.0, el gestor de colas comprueba la contraseña además del nombre de usuario.

Utilización de la fábrica de conexiones a través de una búsqueda indirecta

Al escribir una aplicación empresarial, si el nombre JNDI de la fábrica de conexiones no se conoce, o si la aplicación se va a instalar en servidores de aplicaciones diferentes utilizando una fábrica de conexiones distinta, con un nombre JNDI diferente (en función del servidor de aplicaciones en el que se está instalada), se puede buscar la fábrica de conexiones utilizando una referencia de recurso. Esto se puede realizar a través de una búsqueda indirecta.

Ejemplo

En lugar de buscar directamente la fábrica de conexiones utilizando `jms/myCF`, una aplicación empresarial contiene una referencia de recursos que tiene el nombre JNDI local de: `jms/myResourceReferenceCF`.

Para utilizar este nombre JNDI, la aplicación se conecta al repositorio JNDI del servidor de aplicaciones, de la misma forma que si la aplicación estuviera realizando una búsqueda directa:

```
InitialContext ctx = new InitialContext();
```

En lugar de identificar `jms/myCF` directamente, ahora la aplicación identifica el nombre JNDI de la referencia de recurso:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

Necesita el prefijo `java:comp/env` para el nombre JNDI local, para indicar al servidor de aplicaciones que la aplicación empresarial está realizando una búsqueda indirecta.

Cuando se despliega la aplicación, el usuario correlaciona el nombre JNDI de la referencia de recursos `jms/myResourceReferenceCF` con el nombre JNDI de la fábrica de conexiones que ya ha creado la aplicación: `jms/myCF`.

Cuando se ejecuta la aplicación, busca una fábrica de conexiones JMS utilizando el nombre JNDI, en el que se correlaciona el servidor de aplicaciones: `jms/myCF`. Esta fábrica de conexiones es utilizada después por la aplicación para crear un IBM MQ.

Alias de autenticación y búsqueda indirecta

Una referencia de recursos también permite que se definan propiedades adicionales, que modifican el comportamiento de la fábrica de conexiones proporcionada. Una de las propiedades de una referencia de recurso es **res-auth**. El valor de esta propiedad especifica si la aplicación de empresa debe utilizar el alias de autenticación de la fábrica de conexiones con la que se correlaciona la referencia de recursos al crear una conexión con IBM MQ (si se ha definido un alias de autenticación), o si la aplicación está especificando su propio nombre de usuario y contraseña.

El valor predeterminado de esta propiedad es *Aplicación*. Esto significa que el nombre de usuario y la contraseña que se han pasado al gestor de colas, cuando se crea una conexión JMS, son determinados por la propia aplicación. No se utiliza el alias de autenticación de la fábrica de conexiones con la que está correlacionada la referencia de recursos.

Las aplicaciones pueden crear conexiones JMS utilizando uno de los métodos siguientes:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Si una aplicación utiliza `ConnectionFactory.createConnection()`, y **res-auth** se establece en *Aplicación*, la identidad de usuario predeterminado se pasa a IBM MQ. Este es la combinación de nombre de usuario y contraseña que ha iniciado el servidor de aplicaciones donde se está ejecutando la aplicación empresarial.

Si una aplicación utiliza `ConnectionFactory.createConnection(String username, String password)` y **res-auth** se establece en *Aplicación*, el nombre de usuario y la contraseña pasados al método se envían a IBM MQ.

Para poder utilizar el alias de autenticación definido en la fábrica de conexiones con la que se correlaciona la referencia de recursos al crear una conexión, tendrá que establecer la propiedad **res-auth** en el valor *Contenedor*. Cuando una aplicación crea una conexión JMS, se utilizan los detalles del alias de autenticación, incluso aunque la llamada `createConnection` especifique un nombre de usuario y una contraseña.

Sustitución del alias de autenticación al utilizar una búsqueda indirecta

Si una aplicación utiliza una referencia de recursos que tiene la propiedad **res-auth** establecida en *Contenedor*, puede sustituir el alias de autenticación que se utiliza cuando se crean conexiones JMS.

Para sustituir el alias de autenticación. la referencia de recursos debe incluir una propiedad adicional llamada **authDataAlias**, que se correlaciona con un alias de autenticación existente que ya se ha creado en el entorno del servidor de aplicaciones en el que se desplegará la aplicación. Puede especificar esta propiedad en cualquier referencia de recursos que se haya creado utilizando las herramientas de Rational proporcionadas por IBM.

Mediante este método, puede utilizar un alias de autenticación diferente al utilizar una fábrica de conexiones JMS que se haya buscado indirectamente. Si el alias de autenticación especificado no existe, se puede especificar uno nuevo después de que se haya instalado la aplicación empresarial. Si desea más información, consulte *Referencias de recursos* en la documentación del producto WebSphere Application Server.

Información relacionada para WebSphere Application Server 8.5.5
Referencias de recursos
Información relacionada para WebSphere Application Server 8.0
Referencias de recursos
Información relacionada para WebSphere Application Server 7.0
Referencias de recursos

Equilibrio de carga de trabajo para los beans controlados por mensaje cuando se utilizan clústeres de WebSphere Application Server

Al utilizar aplicaciones de bean controlado por mensaje desplegadas en un clúster WebSphere Application Server 7.0 y WebSphere Application Server 8.0, y configuradas para ejecutarse en la modalidad normal del proveedor de mensajería IBM MQ, uno de los miembros del clúster procesa la mayoría de los mensajes. Puede equilibrar la carga de trabajo de los miembros del clúster para poder distribuir el proceso de mensajes entre más de un miembro de clúster.

IBM MQ incluye una característica denominada **Asynchronous consume**, que permite a las aplicaciones consumir mensajes de forma asíncrona desde una cola utilizando las API denominadas **MQCB** y **MQCTL**.

Las aplicaciones de bean controlado por mensaje que se ejecutan dentro de WebSphere Application Server 7.0 y WebSphere Application Server 8.0, que utilizan la modalidad normal de proveedor de mensajería IBM MQ utilizarán automáticamente esta característica. Cuando se inician las aplicaciones, configurarán un consumidor asíncrono en el destino de JMS que se ha configurado para la supervisión llamando a **MQCB**. Después se llama a la API **MQCTL** para indicar que la aplicación está preparada para recibir mensajes del destino JMS.

Cuando las aplicaciones de bean controlado por mensaje se han desplegado en un clúster de WebSphere Application Server, cada miembro del clúster configurará un consumidor asíncrono para el destino JMS que está supervisando el bean controlado por mensaje en búsqueda de mensajes. El gestor de colas IBM MQ que aloja el destino JMS es el responsable de notificar al miembro del clúster cuando hay un mensaje adecuado en el destino JMS para su proceso.

Cuando WebSphere Application Server se conecta a un gestor de colas de IBM MQ , los mensajes que llegan a un destino de JMS se distribuirán de forma más uniforme a todos los consumidores asíncronos que se han registrado en ese destino de JMS . Para las aplicaciones de bean controlado por mensaje desplegadas dentro de un clúster WebSphere Application Server 7.0 y WebSphere Application Server 8.0, esto significa que los mensajes se distribuirán de forma más uniforme entre los miembros de clúster.

Tareas relacionadas
Configuración de la propiedad JMS **PROVIDERVERSION**

Utilización del paquete de cabeceras de IBM MQ

El paquete de cabeceras de IBM MQ proporciona un conjunto de interfaces y clases de ayuda que puede utilizar para manipular las cabeceras IBM MQ de un mensaje. Normalmente, se utiliza el paquete de cabeceras de IBM MQ porque se desea realizar los servicios administrativos utilizando el servidor de mandatos (utilizando mensajes de formato de mandato programable, PCF).

Acerca de esta tarea

El paquete de cabeceras de IBM MQ se encuentra en los paquetes `com.ibm.mq.headers` y `com.ibm.mq.headers.pcf`. Puede utilizar este recurso para ambas API alternativas que proporciona IBM MQ para su uso en aplicaciones Java:

- IBM MQ classes for Java (conocidas también como IBM MQ Base Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, conocidas también como IBM MQ JMS).

Las aplicaciones IBM MQ Base Java suelen manipular objetos MQMessage y las clases de soporte de cabecera pueden interactuar directamente con estos objetos, puesto que tienen una comprensión nativa de las interfaces de IBM MQ Base Java.

En IBM MQ JMS, la carga útil de un mensaje normalmente es una serie o un objeto de matriz de bytes, que se puede manipular con las corrientes de datos DataInput y DataOutput. El paquete de cabeceras de IBM MQ se puede usar para interactuar con estas corrientes de datos y es apto para manipular cualquier mensaje MQ enviado y recibido por una aplicación IBM MQ JMS.

Por lo tanto, aunque el paquete de cabeceras de IBM MQ contenga referencias al paquete base de IBM MQ Java, también se ha diseñado para ser utilizado en aplicaciones IBM MQ JMS y es apto para ser utilizado en entornos Java Platform, Enterprise Edition (Java EE).

Una forma habitual en la que podría utilizar el paquete de cabeceras de IBM MQ consiste en manipular los mensajes de administración en formato de mandato programable (PCF), por ejemplo, por cualquiera de las razones siguientes:

- Para acceder a los detalles sobre un recurso IBM MQ.
- Para supervisar la profundidad de una cola.
- Para inhibir el acceso a una cola.

Mediante el uso de mensajes PCF con la API IBM MQ JMS, este tipo de administración de recursos centrada en las aplicaciones se puede realizar desde las aplicaciones Java EE sin tener que recurrir al uso de la API base IBM MQ Java .

Procedimiento

- Para utilizar el paquete de cabeceras de IBM MQ para manipular las cabeceras de un mensaje para IBM MQ classes for Java, consulte “Uso con IBM MQ classes for Java” en la página 533.
- Para utilizar el paquete de cabeceras de IBM MQ para manipular las cabeceras de un mensaje para IBM MQ classes for JMS, consulte “Uso con IBM MQ classes for JMS” en la página 534.

Uso con IBM MQ classes for Java

Normalmente, las aplicaciones IBM MQ classes for Java manipulan objetos MQMessage y las clases de soporte de cabecera pueden interactuar directamente con estos objetos, ya que comprenden de forma nativa las interfaces IBM MQ classes for Java.

Acerca de esta tarea

IBM MQ proporciona algunas aplicaciones de ejemplo que ilustran cómo usar el paquete IBM MQ Headers con el API IBM MQ Base Java (IBM MQ classes for Java).

Los ejemplos muestran dos cosas:

- Cómo crear un mensaje PCF para llevar a cabo una acción administrativa y analizar el mensaje de respuesta.
- Cómo enviar este mensaje PCF utilizando IBM MQ classes for Java.

En función de la plataforma que esté utilizando, estos ejemplos se instalan bajo el directorio `pcf` en el directorio `samples` o `tools` de la instalación de IBM MQ (consulte “Directorios de instalación para IBM MQ classes for Java” en la página 364).

Procedimiento

1. Cree un mensaje PCF para llevar a cabo una acción administrativa y analizar el mensaje de respuesta.
2. Envíe este mensaje PCF utilizando IBM MQ classes for Java.

Conceptos relacionados

“Manejo de cabeceras de mensaje de IBM MQ con IBM MQ classes for Java” en la página 392

Se proporcionan clases de Java que representan distintos tipos de cabecera de mensaje. También se proporcionan dos clases auxiliares.

"Manejo de mensajes PCF con IBM MQ classes for Java" en la página 398
Se proporcionan clases Java para crear y analizar mensajes estructurados PCF, facilitar el envío de solicitudes PCF y la recopilación de respuestas PCF.

Uso con IBM MQ classes for JMS

Para utilizar las cabeceras IBM MQ con IBM MQ classes for JMS, realice los mismos pasos básicos que para IBM MQ classes for Java. El mensaje PCF puede crearse y la respuesta analizarse exactamente de la misma manera con el paquete IBM MQ Headers y usando el mismo código de ejemplo de IBM MQ classes for Java.

Acerca de esta tarea

Para enviar un mensaje PCF utilizando la API IBM MQ, la carga útil del mensaje se debe escribir en un mensaje de bytes de JMS y se debe enviar utilizando las API estándar de JMS. La única consideración es que el mensaje no debe contener una RFH2 de JMS ni cualquier otra cabecera con valores específicas en MQMD.

Para enviar un mensaje PCF, siga los pasos siguientes. La forma en que se crea el mensaje PCF, y en que se extrae la información del mensaje de respuesta es la misma que para IBM MQ classes for Java (consulte "Uso con IBM MQ classes for Java" en la página 533).

Procedimiento

1. Cree un destino de cola de JMS que represente el SYSTEM.ADMIN.COMMAND.QUEUE.

   Las aplicaciones de IBM MQ JMS envían los mensajes PCF al SYSTEM.ADMIN.COMMAND.QUEUEy necesita acceso a un objeto Destino JMS que representa esta cola. El destino ha de tener configuradas las siguientes propiedades:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Si está utilizando WebSphere Application Server, debe definir estas propiedades como propiedades personalizadas en el destino.

   Para crear el destino de forma programática en una aplicación, utilice el código siguiente:

   ```
   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. Convierta un mensaje PCF en un mensaje de bytes JMS que contenga los valores MQMD correctos.

   Se debe crear un mensaje de bytes JMS y se debe escribir el mensaje PCF en el mismo. Hay que crear una cola de respuestas, pero no es necesario que tenga una configuración concreta.

   El fragmento de código de ejemplo siguiente muestra cómo crear un mensaje de bytes JMS y escribir un objeto com.ibm.mq.headers,pcf.PCFMessage en el mismo. El objeto PCFMessage (pcfCmd) se ha creado previamente utilizando el paquete de cabeceras de IBM MQ. (Tenga en cuenta que el paquete para cargar el mensaje PCFMessage es 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. Envíe el mensaje y reciba la respuesta utilizando las API JMS estándar.
4. Convierta el mensaje de respuesta en un mensaje PCF para su procesamiento.

   Para recuperar el mensaje de respuesta y procesarlo como un mensaje PCF, utilice el código siguiente:

```
 // 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);
```

Conceptos relacionados

“Mensajes de JMS” en la página 149
Los mensajes de JMS se componen de una cabecera, propiedades y un cuerpo. JMS define cinco tipos de cuerpo de mensaje.

IBM i Configuración de IBM MQ en IBM i con Java y JMS

Esta serie de temas proporciona una descripción general de cómo se configura y prueba IBM MQ con Java y JMS en IBM i utilizando mandatos CL o el entorno qshell.

Nota:

- A partir de IBM MQ 8.0, `ldap.jar`, `jndi.jar` y `jta.jar` forman parte del JDK.
- V 9.3.0 JM 3.0 V 9.3.0 A partir de IBM MQ 9.3.0, Jakarta Messaging 3.0 está soportado para desarrollar nuevas aplicaciones. IBM MQ 9.3.0 sigue dando soporte a JMS 2.0 para las aplicaciones existentes. No está soportado utilizar tanto la API de Jakarta Messaging 3.0 como la API de JMS 2.0 en la misma aplicación. Para obtener más información, consulte Utilización de clases de IBM MQ para JMS/Jakarta Messaging.

Uso de comandos CL

La variable CLASSPATH que ha establecido se utiliza para realizar pruebas con MQ base Java, JMS con JNDI y JMS sin JNDI.

Si no se utiliza un archivo `.profile` en el directorio `/home/Userprofile`, habrá que configurar las siguientes variables de entorno a nivel de sistema. Se puede comprobar si están configuradas con el comando **WRKENVVAR**.

1. Para ver las variables de entorno de todo el sistema, emita el comando: **WRKENVVAR LEVEL(*SYS)**
2. Para ver las variables de entorno específicas de su trabajo, emita el comando: **WRKENVVAR LEVEL(*JOB)**

3. Si no se ha establecido CLASSPATH, emita el mandato siguiente:

JM 3.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. Si la variable QIBM_MULTI_THREADED no está definida, ejecute este comando:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Si la variable QIBM_USE_DESCRIPTOR_STDIO no está definida, ejecute este comando:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Si la variable QSH_REDIRECTION_TEXTDATA no está definida, ejecute este comando:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Uso del entorno qshell

Si utiliza el entorno QSHELL, puede configurar un `.profile` en el directorio `/home/Userprofile`. Para obtener más información, consulte la documentación del intérprete de Qshell (qsh).

Especifique lo siguiente en el archivo `.profile`. Tenga en cuenta que la sentencia CLASSPATH tiene que estar en una única línea, o separarse en distintas líneas utilizando el carácter \ tal como se muestra.

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
```

```
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

Asegúrese de que la biblioteca QMQMJAVA esté en la lista de bibliotecas emitiendo el comando **DSPLIBL**.

Si la biblioteca QMQMJAVA no aparece en la lista, añádala con este comando: **ADDLIBLE LIB(QMQMJAVA)**

IBM i Pruebas de IBM MQ en IBM i con Java

Cómo probar IBM MQ con Java utilizando el programa de ejemplo MQIVP.

Prueba del IBM MQ base con Java

Lleve a cabo el procedimiento siguiente:

1. Verifique que el gestor de colas se ha iniciado y que el estado del gestor de colas está ACTIVO, emitiendo el mandato siguiente:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Verifique que el canal de conexión de servidor JAVA.CHANNEL se ha creado emitiendo el mandato siguiente:

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. Si JAVA.CHANNEL no existe, emita este mandato:

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. Verifique que el proceso de escucha del gestor de colas se está ejecutando para el puerto 1414 o el puerto que esté utilizando, emitiendo el mandato **WRKMQMLSR**.

   a. Si no se ha iniciado ningún proceso de escucha para el gestor de colas, emita este mandato:

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

Ejecución del programa de prueba de ejemplo MQIVP

1. Inicie el qshell desde la línea de mandatos, emitiendo el mandato STRQSH
2. Verifique que se ha establecido la variable CLASSPATH correcta emitiendo el mandato **export** y luego emita el mandato **cd** como se indica a continuación:

   ```
   cd /qibm/proddata/mqm/java/samples/wmqjava/samples
   ```

3. Ejecute el programa **java** emitiendo este mandato:

   ```
   java MQIVP
   ```

Puede pulsar la tecla INTRO cuando se le solicite:

- Tipo de conexión
- Dirección IP
- Nombre del gestor de colas

para utilizar los valores predeterminados. Esto verifica los enlaces del producto, que se pueden encontrar en la biblioteca QMQMJAVA.

Recibirá unos datos de salida similares al ejemplo siguiente. Observe que la declaración de copyright depende de la versión del producto que esté utilizando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

Pruebas de la conexión de cliente IBM MQ Java

Debe especificar:

- Tipo de conexión
- Dirección IP
- Puerto
- Canal de conexión de servidor
- Gestor de colas

Recibirá unos datos de salida similares al ejemplo siguiente. Observe que la declaración de copyright depende de la versión del producto que esté utilizando.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i Pruebas de IBM MQ en IBM i con JMS

Cómo probar IBM MQ con JMS con y sin JNDI

Prueba de JMS sin JNDI utilizando el programa de ejemplo IVTRun

Lleve a cabo el procedimiento siguiente:

1. Verifique que el gestor de colas se ha iniciado y que el estado del gestor de colas está ACTIVO, emitiendo el mandato siguiente:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Inicie el qshell, desde la línea de mandatos, emitiendo el mandato **STRQSH**.
3. Utilice el mandato **cd** para cambiar de directorio, de esta manera:

```
cd /qibm/proddata/mqm/java/bin
```

4. Ejecute el archivo de script:

```
IVTRun -nojndi [-m qmgrname]
```

Recibirá unos datos de salida similares al ejemplo siguiente. Observe que las declaraciones de copyright dependen de las versiones de los productos que esté utilizando:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

Pruebas de la modalidad de cliente de IBM MQ JMS sin JNDI

Lleve a cabo el procedimiento siguiente:

1. Verifique que el gestor de colas se ha iniciado y que el estado del gestor de colas está ACTIVO, emitiendo el mandato siguiente:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Verifique que se ha creado el canal de conexión del servidor, emitiendo el mandato siguiente:

```
WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
MQMNAME(QMGRNAME)
```

3. Verifique que se ha iniciado el proceso de escucha para el puerto correcto, emitiendo el mandato **WRKMQMLSR**
4. Inicie el qshell, desde la línea de mandatos, emitiendo el mandato **STRQSH**.
5. Verifique que la variable CLASSPATH sea correcta, emitiendo el mandato **export**.
6. Utilice el mandato **cd** para cambiar de directorio, de esta manera:

```
cd /qibm/proddata/mqm/java/bin
```

7. Ejecute el archivo de script:

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

Recibirá unos datos de salida similares al ejemplo siguiente. Observe que las declaraciones de copyright dependen de las versiones de los productos que esté utilizando.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
```

```
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Pruebas de IBM MQ JMS con JNDI

Verifique que el gestor de colas se ha iniciado y que el estado del gestor de colas está ACTIVO, emitiendo el mandato siguiente:

```
WRKMQM MQMNAME(QMGRNAME)
```

Utilización del script de prueba de ejemplo IVTRun

Lleve a cabo el procedimiento siguiente:

1. Realice los cambios adecuados en el archivo `JMSAdmin.config`. Para editar este archivo, utilice el mandato **EDTF** (Edit File) desde una línea de mandatos de IBM i

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. Para utilizar LDAP para Weblogic, elimine el símbolo de comentario en:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
      ```

   b. Para utilizar LDAP para WebSphere Application Server, elimine el símbolo de comentario en:

      ```
      INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
      ```

   c. Para probar el sistema de archivos, elimine el símbolo de comentario en:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
      ```

   d. Asegúrese de que ha seleccionado el PROVIDER_URL correcto, eliminando el símbolo de comentario en la línea adecuada.

   e. Convierta en comentario todas las demás líneas utilizando el símbolo #.

   f. Cuando haya completado todos los cambios, pulse **F2=Save** y **F3=Exit**.

2. Inicie el qshell, desde la línea de mandatos, emitiendo el mandato **STRQSH**.
3. Verifique que la variable CLASSPATH sea correcta, emitiendo el mandato **export**.
4. Utilice el mandato **cd** para cambiar de directorio, de esta manera:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

5. Inicie el script **IVTSetup** para crear los objetos administrados (*MQQueueConnectionFactory* y *MQQueue*), emitiendo el mandato **IVTSetup**.
6. Ejecute el script IVTRun emitiendo el mandato siguiente:

   ```
   IVTRun -url providerURL [-icf initCtxFact]
   ```

Recibirá unos datos de salida similares al ejemplo siguiente. Observe que las declaraciones de copyright dependen de las versiones de los productos que esté utilizando.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
```

```
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Desarrollo de aplicaciones Java utilizando un repositorio de Maven

Al desarrollar una aplicación Java para IBM MQ, utilizando un repositorio de Maven para instalar dependencias automáticamente, no tendrá que instalar nada explícitamente antes de utilizar interfaces IBM MQ.

Repositorio central de Maven

Maven es una herramienta para crear aplicaciones y, también, proporciona un repositorio para contener artefactos a los que desea que acceda la aplicación.

El repositorio de Maven (o repositorio central) tiene una estructura que permite que los archivos como, por ejemplo, archivos JAR tengan versiones distintas que se descubren después fácilmente con un mecanismo de denominación bien conocido. A continuación, las herramientas de creación pueden utilizar

estos nombres para extraer dinámicamente las dependencias de la aplicación. En la definición de la aplicación, que, cuando se utiliza Maven como herramienta de compilación, se denomina archivo POM, se denominan las dependencias y el proceso de compilación sabe qué hacer a partir de ahí.

Archivos cliente de IBM MQ

Las copias de las interfaces de cliente de IBM MQ Java están disponibles en el repositorio central bajo `com.ibm.mq GroupId`. Puede encontrar el archivo `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) y el archivo `com.ibm.mq.allclient.jar` (JMS 2.0). Estos archivos se utilizan normalmente para programas autónomos. También puede encontrar el archivo `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) y el archivo `wmq.jmsra.rar` (JMS 2.0), que se utiliza en los servidores de aplicaciones Java EE). El `jakarta.client.jar` y el `allclient.jar` contienen el IBM MQ classes for JMS y el IBM MQ classes for Java.

Importante: El uso del formato Apache Maven Assembly Plugin *jar-with-dependencias* para crear una aplicación que incluya el archivo JAR reubicable IBM MQ no está soportado.

En un archivo `pom.xml` procesado por el mandato maven, añada dependencias para estos archivos JAR, tal como se muestra en los ejemplos siguientes:

- V 9.3.0 JM 3.0 V 9.3.0 Para mostrar la relación entre el código de aplicación y `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Para mostrar la relación entre el código de aplicación y `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- V 9.3.0 JM 3.0 V 9.3.0 Para utilizar el adaptador de recursos Jakarta EE:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 Para utilizar el adaptador de recursos de JMS 2.0 Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Para ver un ejemplo de un proyecto simple en Eclipse para ejecutar un proyecto JMS , consulte el artículo de IBM Developer Desarrollo de aplicaciones Java para MQ que acaba de ser más fácil con Maven.

Desarrollo de aplicaciones C++

IBM MQ proporciona clases C++ equivalentes a los objetos IBM MQ y algunas clases equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

IBM WebSphere MQ 7.0, las mejoras en las interfaces de programación de IBM MQ no se aplican a las clases C++.

IBM MQ C++ proporciona las características siguientes:

- Inicialización automática de estructuras de datos de IBM MQ.
- Conexión puntual del gestor de colas y apertura de colas.
- Cierre de colas implícito y desconexión del gestor de colas.
- Recepción y transmisión de la cabecera de mensajes no entregados.
- Recepción y transmisión de la cabecera del puente de IMS.
- Recepción y transmisión de la cabecera del mensaje de referencia.
- Recepción de mensajes de desencadenantes.
- Recibo y transmisión de la cabecera de CICS bridge.
- Recepción y transmisión de la cabecera de trabajo.
- Definición de canal de cliente.

Los diagramas de clases Booch siguientes muestran que todas las clases son en líneas generales paralelas a las entidades de IBM MQ en la MQI procedimental (por ejemplo, utilizando C) que tienen descriptores o estructuras de datos. Todas las clases heredan de la clase ImqError (véase la clase C++ ImqError), que permite que se asocie una condición de error a cada objeto.

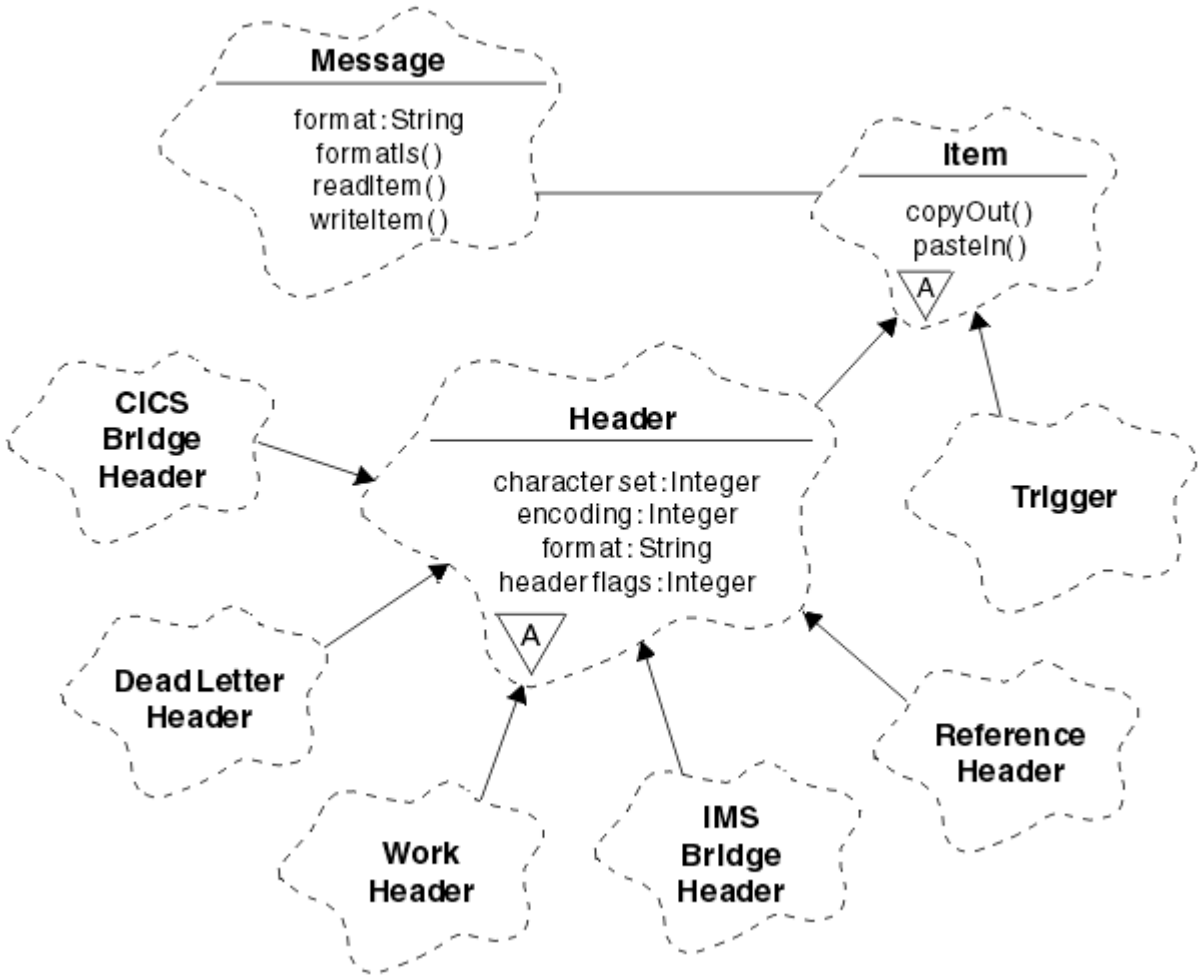


Figura 50. Clases C++ de IBM MQ (manejo de elementos)

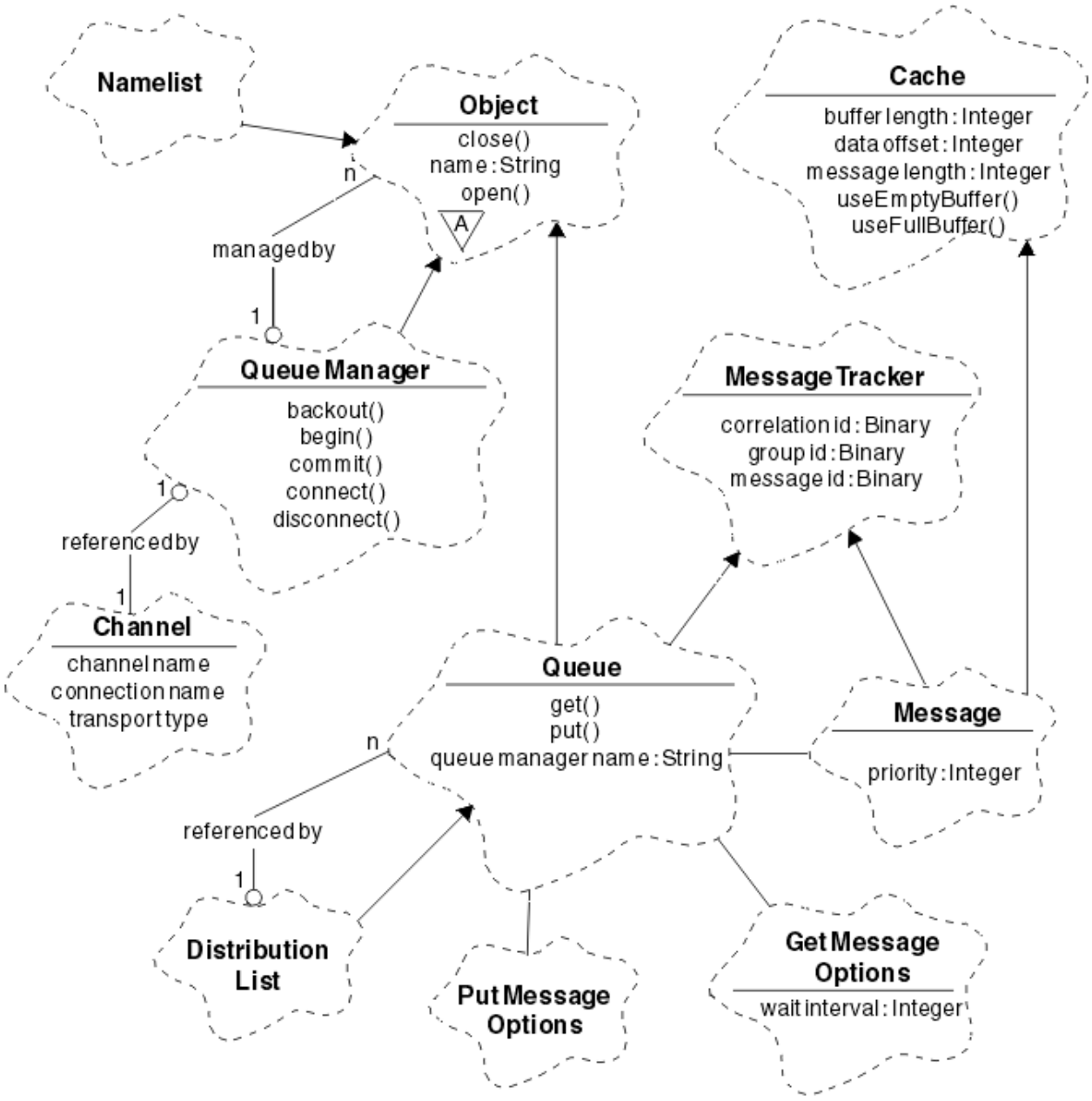


Figura 51. Clases C++ de IBM MQ (gestión de colas)

Para interpretar diagramas de clase Booch de forma correcta, tenga en cuenta los siguientes convenios:

- Los métodos y los atributos destacables se muestran debajo del nombre *class*.
- Un pequeño triángulo dentro de una nube indica una *clase abstracta*.
- La *herencia* viene indicada por una flecha en la clase padre.
- Una línea no decorada entre nubes indica una *relación cooperativa* entre clases.
- Una línea decorada con un número señala una *relación referencial* entre dos clases. El número indica el número de objetos que pueden participar en una relación determinada en cualquier momento.

Las clases y tipos de datos siguientes se utilizan en las firmas de métodos de C++ de las clases de gestión de colas (consulte Figura 51 en la página 545) y las clases de gestión de elementos (consulte Figura 50 en la página 544):

- La clase ImqBinary (consulte la clase C++ ImqBinary), que encapsula matrices de bytes como MQBYTE24.

- El tipo de datos ImqBoolean, que se define como **typedef unsigned char ImqBoolean**.
- La clase ImqString (consulte la clase C++ ImqString), que encapsula matrices de caracteres como MQCHAR64.

Las entidades con estructuras de datos se incluyen dentro de las clases de objeto adecuadas. Se accede mediante métodos a los campos de estructura de datos individuales (consulte Referencia cruzada de C++ y MQI).

Las entidades con manejadores se encuentran bajo la jerarquía de la clase ImqObject (consulte la clase C++ ImqObject) y proporcionan interfaces encapsuladas a la MQI. Los objetos de estas clases presentan un comportamiento inteligente que puede reducir el número de invocaciones de método necesarios relativos a la interfaz de cola de mensajes de procedimiento. Por ejemplo, puede establecer y descartar las conexiones del gestor de colas, según sea necesario, o puede abrir una cola con las opciones adecuadas y, a continuación, cerrarla.

La clase ImqMessage (consulte la clase C++ ImqMessage) encapsula la estructura de datos MQMD y también actúa como punto de apoyo para los *elementos* y los datos de usuario (consulte "Lectura de mensajes en C++" en la página 555) proporcionando recursos de almacenamiento intermedio en memoria caché. Puede proporcionar almacenamientos intermedios de longitud fija para datos de usuario y utilizar dicho almacenamiento intermedio muchas veces. La cantidad de datos presentes en el almacenamiento intermedio puede variar de un uso al siguiente. De forma alternativa, el sistema puede proporcionar y gestionar un almacenamiento intermedio de longitud flexible. Tanto el tamaño del almacenamiento intermedio (la cantidad disponible para la recepción de mensajes) como la cantidad realmente utilizada (o el número de bytes para la transmisión o el número de bytes realmente recibidos) pasan a ser consideraciones importantes.

Conceptos relacionados

Visión general técnica

"Programas de ejemplo C++" en la página 546
Se proporcionan cuatro programas de ejemplo que muestran cómo obtener y transferir mensajes.

"consideraciones relativas a C++" en la página 550
Esta colección de temas detalla los aspectos del uso de C++ y las convenciones que hay que tener en cuenta al desarrollar aplicaciones que usan la interfaz de colas de mensajes (MQI).

"Preparación de datos de mensaje en C++" en la página 554
Los datos de mensajes se preparan en un almacenamiento intermedio, que puede suministrar el sistema o la aplicación. Hay ventajas para cualquiera de los dos métodos. Se proporcionan ejemplos de utilización de un almacenamiento intermedio.

"Desarrollo de aplicaciones para IBM MQ" en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

Referencia relacionada

"Creación de programas IBM MQ en C++" en la página 561
El URL de los compiladores soportados está listado junto con los mandatos para utilizar para compilar, enlazar y ejecutar programas C++ y ejemplos en plataformas IBM MQ.

Referencia cruzada de C++ y MQI

Clases C++ de IBM MQ

Programas de ejemplo C++

Se proporcionan cuatro programas de ejemplo que muestran cómo obtener y transferir mensajes.

Los programas de ejemplo son:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)

- DPUT (imqdput.cpp)

Los programas de ejemplo se encuentran en los directorios que se muestran en la Tabla 73 en la página 547.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Tabla 73. Ubicación de los programas de ejemplo

Entorno	Directorio que contiene el origen	Directorio que contiene programas programas
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
V 9.3.5 AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (Consulte la nota “1” en la página 547)
IBM i	`/QIBM/ProdData/mqm/samp/`	(Consulte la nota “2” en la página 547)
Linux	*MQ_INSTALLATION_PATH*/samp	Ninguna
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn (Consulte la nota “3” en la página 547)
z/OS	thlqual.SCSQCPPS	

Notas:

1. V 9.3.5 AIX Los programas compilados utilizando el compilador XLC 17 se encuentran en la carpeta "`ca`", mientras que los programas compilados utilizando el compilador XLC 16 se encuentran en la carpeta "`ia`".
2. IBM i Los programas compilados con el compilador ILE C++ para IBM i están en la biblioteca QMQM. Los archivos de origen se encuentran en `/QIBM/ProdData/mqm/samp`.
3. Windows Los programas compilados con Microsoft Visual Studio Visual Studio están en *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Para obtener más información sobre estos compiladores, consulte “Compilación de programas C++ en Windows” en la página 567.

Programa de ejemplo HELLO WORLD (imqwrld.cpp)

Este programa C++ de ejemplo muestra cómo transferir y obtener un datagrama normal (estructura C) utilizando la clase ImqMessage.

Este programa muestra cómo transferir y obtener un datagrama normal (estructura C) utilizando la clase ImqMessage. Este ejemplo utiliza varios invocaciones de métodos, aprovechando invocaciones de métodos implícitas, como **open**, **close** y **disconnect**.

En todas las plataformas excepto z/OS

Si utiliza una conexión de servidor con IBM MQ, siga uno de los procedimientos siguientes:

- Para utilizar la cola predeterminada existente, SYSTEM.DEFAULT.LOCAL.QUEUE, ejecute el programa **`imqwrlds`** sin pasar ningún parámetro

- Para utilizar una cola temporal asignada dinámicamente, ejecute **imqwrlds** pasando el nombre de la cola de modelo predeterminada, SYSTEM.DEFAULT.MODEL.QUEUE.

Si utiliza una conexión de cliente con IBM MQ, siga uno de los procedimientos siguientes:

- Configure la variable de entorno MQSERVER (consulte *MQSERVER* para obtener más información) y ejecute **imqwrldc**, o
- Ejecute **imqwrldc** pasando como parámetros **queue-name**, **queue-manager-name** y **channel-definition**, donde **channel-definition** típicamente puede ser *nombre_host* (1414)

En z/OS

z/OS

Construya y ejecute un trabajo por lotes, utilizando la JCL de ejemplo **imqwrldr**.

Consulte Lote z/OS, Lote RRS y CICS para obtener más información.

Código de ejemplo

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
```

```
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

Programas de ejemplo SPUT (imqsput.cpp) y SGET (imqsget.cpp)

Estos programas C++ colocan mensajes en una cola con nombre y recuperan mensajes de una cola con nombre.

Estos ejemplos muestran el uso de las clases siguientes:

- ImqError (consulte Clase C++ ImqError)
- ImqMessage (consulte Clase C++ ImqMessage)
- ImqObject (consulte Clase C++ ImqObject)
- ImqQueue (consulte Clase C++ ImqQueue)
- ImqQueueManager (consulte Clase C++ ImqQueueManager)

Siga las instrucciones adecuadas para ejecutar los programas.

En todas las plataformas excepto z/OS

1. Ejecuta **imqsputs** *nombre_cola*.
2. Escriba las líneas de texto en la consola. Estas líneas se colocan como mensajes en la cola especificada.
3. Especifique una línea nula para finalizar la entrada.
4. Ejecute **imqsget** *nombre_cola* para recuperar todas las líneas y visualizarlas en la consola.

z/OS Consulte "Creación de programas C++ en z/OS Batch, RRS Batch y CICS" en la página 569 para obtener más información.

En z/OS

z/OS

1. Construya y ejecute un trabajo por lotes utilizando el JCL de ejemplo **imqsputr**. Los mensajes se leen en el conjunto de datos SYSIN.
2. Construya y ejecute un trabajo por lotes utilizando el JCL de ejemplo **imqsgetr**. Los mensajes se recuperan de la cola y se envían al conjunto de datos SYSPRINT.

Programa de ejemplo DPUT (imqdput.cpp)

Este programa de ejemplo de C++ coloca los mensajes en una lista de distribución que consta de dos colas.

DPUT muestra el uso de la clase ImqDistributionList (consulte ImqDistributionList C++ class). Este ejemplo no es válido en z/OS.

1. Ejecute **imqdputs** *queue-name-1 queue-name-2* para colocar mensajes en las dos colas con nombre.
2. Ejecute **imqsgets** *queue-name-1* y **imqsgets** *queue-name-2* para recuperar mensajes de dichas colas.

consideraciones relativas a C++

Esta colección de temas detalla los aspectos del uso de C++ y las convenciones que hay que tener en cuenta al desarrollar aplicaciones que usan la interfaz de colas de mensajes (MQI).

Archivos de cabecera de C++

Los archivos de cabecera se proporcionan como parte de la definición de MQI, para ayudarle a escribir los programas de aplicación de IBM MQ en el lenguaje C++.

Estos archivos de cabecera se resumen en la tabla siguiente.

Tabla 74. Archivos de cabecera C/C++

Nombre de archivo	Contenido
IMQI.HPP	Clases MQI de C++ (incluye CMQC.H e IMQTYPE.H)
IMQTYPE.H	Define el tipo de datos **ImqBoolean**

Tabla 74. Archivos de cabecera C/C++ (continuación)	
Nombre de archivo	**Contenido**
CMQC.H	Estructuras de datos de MQI y constantes de manifiesto

Para mejorar la portabilidad de aplicaciones, codifique el nombre del archivo de cabecera en minúsculas en la directiva de preprocesador **#include**:

```
#include <imqi.hpp> // C++ classes
```

Métodos y atributos de C++

Los nombres de método incluyen mayúsculas y minúsculas. Se aplican varias consideraciones a los parámetros y a los valores de retorno. Se accede a los atributos utilizando los métodos set y get, según corresponda.

Los parámetros de los métodos que son *const* son solo para la entrada. Los parámetros con firmas que incluyen un puntero (*) o una referencia (&) se pasan por referencia. Los valores de retorno que no incluyen un puntero o una referencia se pasan por valor; en el caso de los objetos devueltos, son entidades nuevas que se convierten en responsabilidad del emisor.

Algunas firmas de método incluyen elementos que toman un valor predeterminado si no se especifican. Dichos elementos siempre están al final de las firmas y están denotados por un signo igual (=); el valor después del signo de igual indica el valor predeterminado que se aplica si se omite el elemento.

Todos los nombres de método de estas clases incluyen mayúsculas y minúsculas, y empiezan por minúscula. Cada palabra, excepto la primera dentro de un nombre de método, empieza por una mayúscula. No se utilizan abreviaturas, a menos que se entienda ampliamente su significado. Las abreviaturas utilizadas incluyen *id* (para la identidad) y *sync* (para la sincronización).

Se accede a los atributos de objeto utilizando los métodos set y get. Un método set empieza por la palabra *set*; un método get no tiene prefijo. Si un atributo es de *solo lectura*, no hay ningún método set.

Los atributos se inicializan en estados válidos durante la construcción de objetos, y el estado de un objeto siempre es coherente.

Tipos de datos en C++

Todos los tipos de datos se definen mediante la sentencia de C **typedef**.

El tipo **ImqBoolean** se define como **unsigned char** en IMQTYPE.H y puede tener los valores TRUE y FALSE. Puede utilizar los objetos de clase **ImqBinary** en lugar de matrices **MQBYTE**, y los objetos de clase **ImqString** en lugar de **char ***. Muchos métodos devuelven objetos en lugar de los punteros de **char** o **MQBYTE** para facilitar la gestión del almacenamiento. Todos los valores de retorno pasan a ser responsabilidad del emisor y, en el caso de un objeto devuelto, el almacenamiento se puede eliminar utilizando la supresión.

Manipulación de series binarias en C++

Las series de datos binarios se declaran como objetos de la clase **ImqBinary**. Los objetos de esta clase pueden copiarse, compararse y establecerse utilizando los conocidos operadores de C. Se proporciona un código de ejemplo.

El siguiente ejemplo de código muestra operaciones en una serie binaria:

```
#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.
...
```

Manipulación de las series de caracteres en C++

Normalmente, los datos de caracteres se devuelven objetos de clase **ImqString**, que se pueden convertir en **char *** utilizando un operador de conversión. La clase ImqString contiene métodos que ayudan a procesar las series de caracteres.

Cuando se aceptan o devuelven los datos de caracteres utilizando los métodos MQI C++, los datos de caracteres siempre terminan en nulos y pueden tener cualquier longitud. No obstante, IBM MQ impone algunos límites que pueden dar lugar a que la información se muestre truncada. Para facilitar la gestión del almacenamiento, frecuentemente los datos de caracteres se devuelven en objetos de clase **ImqString**. Estos objetos se pueden convertir a **char *** utilizando el operador de conversión proporcionado, y se pueden utilizar para fines de *solo lectura* en muchas situaciones en las que se requiere un **char ***.

Nota: El resultado de la conversión **char *** de un objeto de clase **ImqString** puede ser nulo.

Aunque se pueden utilizar las funciones C en **char ***, hay métodos especiales de la clase **ImqString** que son preferibles. La **longitud del operador** ( ) es equivalente a **strlen** y **almacenamiento** ( ) indica la memoria asignada para los datos de caracteres.

Estado inicial de los objetos en C++

Todos los objetos tienen un estado inicial coherente reflejado en sus atributos. Los valores iniciales se definen en las descripciones de clase.

Utilización de C desde C++

Cuando utilice funciones de C desde un programa C++, incluya las cabeceras adecuadas.

El ejemplo siguiente muestra la cabecera `string.h` incluida en un programa C++:

```
extern "C" {
#include <string.h>
}
```

Convenios de anotaciones de C++

Este ejemplo muestra cómo utilizar los métodos de invocación y declarar parámetros.

Este ejemplo de código utiliza los métodos y parámetros **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Declare y utilice los parámetros como se indica a continuación:

```
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( );

...
}
```

Operaciones implícitas en C++

Se pueden realizar varias operaciones de forma implícita, *justo a tiempo*, para cumplir los requisitos previos a la ejecución satisfactoria de un método. Estas operaciones implícitas son conectar, abrir, reabrir, cerrar y desconectar. Se puede controlar la conexión y abrir el comportamiento implícito utilizando atributos de clase.

Conectar

En cualquier método que resulte en cualquier llamada a la MQI (consulte Referencia cruzada de C++ y MQI), se conecta automáticamente un objeto ImqQueueManager.

Abrir

En cualquier método que resulte en una llamada MQGET, MQINQ, MQPUT o MQSET se abre automáticamente objeto ImqObject. Utilice el método **openFor** para especificar uno o más valores de **opción abierta** relevantes.

Reabrir

En cualquier método que resulte en una llamada MQGET, MQINQ, MQPUT o MQSET donde el objeto ya esté abierto, pero las **opciones de apertura** existentes no sean adecuadas para permitir que la llamada MQI sea satisfactoria, se reabre el ImqObject. El objeto se cierra temporalmente utilizando el valor de **opciones de cierre** temporal MQCO_NONE. Utilice el método **openFor** para añadir una**opción de apertura** relevante.

Una reapertura puede ocasionar problemas en determinadas circunstancias:

- Una cola dinámica temporal se destruye cuando se cierra y nunca se puede reabrir.
- Una cola abierta para entrada exclusiva (de forma explícita o predeterminada) podría ser accedida por otros en la ventana de oportunidad durante el cierre y la reapertura.
- Una posición de cursor de examen se pierde al cerrarse una cola. Esta situación no impide el cierre y la reapertura, pero impedirá el uso posterior del cursor mientras no se vuelva a utilizar MQGMO_BROWSE_FIRST.
- El contexto del último mensaje recuperado se ha perdido al cerrar una cola.

Si cualquiera de estas circunstancias se produce o se puede prever, evite volver las reaperturas explícitas estableciendo las correspondientes **opciones de apertura** antes de abrir un objeto (de forma explícita o implícita).

La definición explícita de las **opciones de apertura** en situaciones complejas de manejo de colas mejora el rendimiento y evita los problemas asociados al uso de la reapertura.

Cerrar

Un objeto ImqObject se cierra automáticamente en cualquier punto en el que el estado del objeto ya no sea viable como, por ejemplo, si se ha perdido una referencia de conexión ImqObject o si se destruye un objeto ImqObject.

Desconectar

Un objeto ImqQueueManager se desconecta automáticamente en cualquier punto en el que la conexión ya no sea viable como, por ejemplo, si se pierde una referencia de conexión ImqObject o si se destruye un objeto ImqQueueManager.

Series binarias y de caracteres en C++

La clase ImqString encapsula el formato de datos *char ** tradicional. La clase ImqBinary encapsula la matriz de bytes binarios. Algunos métodos que establecen los datos de caracteres podrían truncar los datos.

Los métodos que establecen los datos (**char ***) de caracteres siempre toman una copia de los datos, pero algunos métodos pueden truncar la copia porque IBM MQ impone ciertos límites.

La clase ImqString (consulte la ImqString C++ class) encapsula el **char *** tradicional y proporciona soporte para:

- Comparación
- Concatenación
- Copia
- Conversión entero-a-texto y texto-a-entero
- Extracción de Token (word)
- Conversión a mayúsculas

La clase ImqBinary (consulte la ImqBinary C++ class) encapsula las matrices de bytes binarios de tamaño arbitrario. En particular, se utiliza para conservar los atributos siguientes:

- **accounting token** (MQBYTE32)
- **connection tag** (MQBYTE128)
- **correlation id** (MQBYTE24)
- **facility token** (MQBYTE8)
- **group id** (MQBYTE24)
- **instance id** (MQBYTE24)
- **message id** (MQBYTE24)
- **message token** (MQBYTE16)
- **transaction instance id** (MQBYTE16)

Donde estos atributos pertenecen a objetos de las clases siguientes:

- ImqCICSBridgeHeader (consulte la ImqCICSBridgeHeader C++ class)
- ImqGetMessageOptions (consulte la ImqGetMessageOptions C++ class)
- ImqIMSBridgeHeader (consulte la ImqIMSBridgeHeader C++ class)
- ImqMessageTracker (consulte la ImqMessageTracker C++ class)
- ImqQueueManager (consulte Clase C++ ImqQueueManager)
- ImqReferenceHeader (consulte la ImqReferenceHeader C++ class)
- ImqWorkHeader (consulte la ImqWorkHeader C++ class)

La clase ImqBinary también proporciona soporte para la comparación y la copia.

Funciones no soportadas en C++

Las clases y los métodos C++ de IBM MQ son independientes de la plataforma IBM MQ. Por lo tanto, es posible que ofrezcan algunas funciones que no estén soportadas en determinadas plataformas.

Si se intenta usar una función en una plataforma en la que no está soportada, dicha función es detectada por IBM MQ, pero no por los enlaces del lenguaje C++. IBM MQ notifica el error al programa, como cualquier otro error de MQI.

Mensajería en C++

En esta colección de temas se detalla cómo preparar, leer y escribir mensajes en C++.

Preparación de datos de mensaje en C++

Los datos de mensajes se preparan en un almacenamiento intermedio, que puede suministrar el sistema o la aplicación. Hay ventajas para cualquiera de los dos métodos. Se proporcionan ejemplos de utilización de un almacenamiento intermedio.

Cuando se envía un mensaje, los datos de mensaje se preparan primero en un almacenamiento intermedio gestionado por un objeto ImqCache (consulte la clase ImqCache C++). Un almacenamiento intermedio está asociado (por herencia) con cada objeto ImqMessage (consulte Clase C++ de ImqMessage): lo puede proporcionar la aplicación (utilizando el método **useEmpEmptyBuffer** o **useFullBuffer**) o automáticamente el sistema. La ventaja de la aplicación que proporciona el almacenamiento intermedio de mensajes es que no es necesario realizar ninguna copia de datos en muchos casos porque la aplicación puede utilizar áreas de datos preparadas directamente. La desventaja es que el almacenamiento intermedio proporcionado es de una longitud fija.

El almacenamiento intermedio se puede reutilizar, y el número de bytes transmitidos puede variarse cada vez, utilizando el método **setMessageLength** antes de la transmisión.

Cuando el sistema lo proporciona automáticamente, el sistema gestiona el número de bytes disponibles, y los datos se pueden copiar en el almacenamiento intermedio de mensajes utilizando, por ejemplo, el método **write** de ImqCache o el método **writeItem** de ImqMessage. El almacenamiento intermedio de mensajes crece según las necesidades. A medida que el búfer crece, no hay pérdida de datos previamente escritos. Un mensaje grande o de varias partes se puede escribir en partes secuenciales.

En los ejemplos siguientes se muestran envíos de mensaje simplificados.

1. Utilizar datos preparados en un almacenamiento intermedio proporcionado por el usuario

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Utilizar datos preparados en un almacenamiento intermedio proporcionado por el usuario, donde el tamaño de almacenamiento intermedio sobrepase el tamaño de los datos

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Copiar datos en un almacenamiento intermedio proporcionado por el usuario

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Copiar datos en un almacenamiento intermedio proporcionado por el sistema

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Copiar datos en un almacenamiento intermedio proporcionado por el sistema utilizando objetos (los objetos establecen el formato del mensaje así como el contenido)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Lectura de mensajes en C++

Un almacenamiento intermedio lo puede proporcionar la aplicación o el sistema. A los datos se puede acceder directamente desde el almacenamiento intermedio o mediante lectura secuencial. Hay una clase equivalente a cada tipo de mensaje. Se proporciona código de ejemplo.

Cuando se reciben datos, la aplicación o el sistema pueden suministrar un almacenamiento intermedio de mensajes adecuado. El mismo almacenamiento intermedio se puede utilizar tanto para la transmisión múltiple como para la recepción múltiple para un objeto ImqMessage determinado. Si el almacenamiento intermedio de mensajes se proporciona automáticamente, crece para dar cabida a la longitud de los datos que se reciban. No obstante, un almacenamiento intermedio de mensaje proporcionado por la aplicación podría no ser suficiente para alojar los datos recibidos. Se producirá truncamiento o error, en función de las opciones utilizadas para la recepción de mensajes.

Se puede acceder a los datos entrantes directamente desde el almacenamiento intermedio de mensajes, en cuyo caso la longitud de datos indica la cantidad total de datos de entrada. De forma alternativa, los datos entrantes se pueden leer secuencialmente desde el almacenamiento intermedio de mensajes. En este caso, el puntero de datos se dirige al siguiente byte de datos de entrada, y el puntero y la longitud de datos se actualizan cada vez que se leen los datos.

Los *Elementos* son partes de un mensaje, todas en el área de usuario del almacenamiento intermedio de mensajes, que se deben procesar secuencialmente y por separado. Aparte de los datos de usuario habituales, un elemento puede ser una cabecera de mensajes no entregados o un mensaje desencadenante. Los elementos siempre están asociados con los formatos de mensaje; los formatos de mensaje **no** siempre están asociados con elementos.

Hay una clase de objeto para cada elemento que corresponde a un formato de mensaje de IBM MQ reconocible. Hay una para una cabecera de mensajes no entregados y otra para un mensaje desencadenante. No hay ninguna clase de objeto para los datos de usuario. Es decir, una vez que los formatos reconocibles se han agotado, el proceso del resto se deja en el programa de aplicación. Las clases para los datos de usuario se pueden escribir especializándose en la clase ImqItem.

El ejemplo siguiente muestra un recibo de mensaje que tiene en cuenta un número de elementos potenciales que pueden preceder a los datos de usuario, en una situación imaginaria. Los datos de usuario que no son de elemento se definen como cualquier cosa que se produce después de los elementos que se pueden identificar. Se utiliza un almacenamiento intermedio automático (el valor predeterminado) para alojar una cantidad arbitraria de datos de mensaje.

```
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.                      */
      ...

    }

  }
}
```

En este ejemplo, FMT_USERCLASS es una constante que representa el nombre de formato de 8 caracteres asociado con un objeto de clase `UserClass`, y está definido por la aplicación.

`UserClass` se deriva de la clase ImqItem (consulte Clase C++ de ImqItem) e implementa los métodos **copyOut** y **pasteIn** virtuales de dicha clase.

Los dos ejemplos siguientes muestran código de la clase ImqDeadLetterHeader (consulte la Clase C++ ImqDeadLetterHeader). El primer ejemplo muestra el código de *escritura* de mensajes encapsulados personalizado.

```
// 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 ;
}
```

El segundo ejemplo muestra el código de *lectura* de mensajes encapsulados personalizado.

```
// 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 ;
}
```

Con un almacenamiento intermedio automático, el almacenamiento es *volátil*. Es decir, los datos de almacenamiento intermedio se pueden mantener en una ubicación física distinta después de cada invocación de método **get**. Por lo tanto, cada vez que se hace referencia a los datos de almacenamiento intermedio, utilice los métodos **bufferPointer** o **dataPointer** para acceder a los datos de los mensajes.

Es posible que quiera que un programa aparte un área fija para recibir datos de mensaje. En tal caso, invoque el método **useEmptyBuffer** antes de usar el método **get**.

El uso de un área fija y no automática limita los mensajes a un tamaño máximo, por lo que es importante tener en cuenta la opción MQGMO_ACCEPT_TRUNCATED_MSG del objeto ImqGetMessageOptions.
Si no se especifica esta opción (valor predeterminado), se puede esperar el código de razón MQRC_TRUNCATED_MSG_FAILED. Si se especifica esta opción, es posible que se pueda esperar el código de razón MQRC_TRUNCATED_MSG_ACCEPTED en función del diseño de la aplicación.

En el ejemplo siguiente se muestra cómo se puede utilizar un área de almacenamiento fija para recibir mensajes:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

En este fragmento de código, siempre se puede hacer referencia directa al almacenamiento intermedio, con *pszBuffer*, al contrario de lo que sucede al usar el método **bufferPointer**. Sin embargo, es mejor utilizar el método **dataPointer** para el acceso para fines generales. La aplicación (no el objeto de clase ImqCache) debe descartar un almacenamiento intermedio definido por el usuario (no automático).

Atención: la especificación un puntero nulo y una longitud cero con **useEmptyBuffer**, no designa un almacenamiento intermedio de longitud fija de longitud cero como cabría esperar. Esta combinación se interpreta como una solicitud para ignorar cualquier almacenamiento intermedio anterior definido por el usuario y, en su lugar, revierte a la utilización de un almacenamiento intermedio automático.

Escritura de un mensaje en la cola de mensajes no entregados en C++

Código de programa de ejemplo para escribir un mensaje en la cola de mensajes no entregados.

Un caso típico de un mensaje de varias partes es uno que contiene una cabecera de mensajes no entregados. Los datos de un mensaje que no se pueden procesar se añaden a la cabecera de mensaje no entregado.

```
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 );
```

Escritura de un mensaje en el puente IMS en C++

Código de programa de ejemplo para escribir un mensaje en el puente IMS.

Los mensajes enviados al puente IBM MQ - IMS pueden utilizar una cabecera especial. A la cabecera del puente IMS le preceden los datos de mensajes normales.

```
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 );
```

Escritura de un mensaje en el CICS bridge en C++

Código de programa de ejemplo para escribir un mensaje en el CICS bridge.

Los mensajes enviados a IBM MQ for z/OS utilizando el CICS bridge requieren una cabecera especial. A la cabecera de CICS bridge le preceden los datos de mensajes normales.

```
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 );
```

Escritura de un mensaje con una cabecera de trabajo en C++

Código de programa de ejemplo para escribir un mensaje destinado a una cola gestionada por el gestor de carga de trabajo de z/OS.

Los mensajes enviados a IBM MQ for z/OS, que están destinados para una cola gestionada por el gestor de la carga de trabajo z/OS, requieren una cabecera especial. El prefijo de la cabecera de trabajo son datos de mensaje habituales.

```
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 );
```

Creación de programas IBM MQ en C++

El URL de los compiladores soportados está listado junto con los mandatos para utilizar para compilar, enlazar y ejecutar programas C++ y ejemplos en plataformas IBM MQ.

Para obtener una lista de los compiladores para cada plataforma soportada y cada versión de IBM MQ, consulte Requisitos del sistema para IBM MQ.

El mandato que tiene que compilar y enlazar al programa C++ de IBM MQ depende de la instalación y de los requisitos. Los ejemplos que aparecen a continuación muestran los mandatos típicos de compilación y enlazado para algunos de los compiladores que utilizan la instalación predeterminada de IBM MQ en una serie de plataformas.

AIX Compilación de programas C++ en AIX

Compile programas C++ de IBM MQ en AIX utilizando el compilador XL C Enterprise Edition.

V 9.3.5 Para obtener más información sobre la correlación diferente de opciones de compilador entre compiladores XLC 16 y XLC 17, consulte Correlación de opciones.

V 9.3.5 Deprecated El soporte para el compilador XL C/C++ para AIX 16 en AIX está en desuso desde IBM MQ 9.3.5.

Cliente

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Aplicación sin hebras de 32 bits

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Aplicación con hebras de 32 bits

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Aplicación sin hebras de 64 bits

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Aplicación con hebras de 64 bits

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.3.5 **Aplicación sin hebras de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **Aplicación con hebras de 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.3.5 **Aplicación sin hebras de 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **Aplicación con hebras de 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

Servidor

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Aplicación sin hebras de 32 bits

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Aplicación con hebras de 32 bits

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Aplicación sin hebras de 64 bits

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Aplicación con hebras de 64 bits

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.3.5 Aplicación sin hebras de 32 bits (XLC 17)

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 Aplicación con hebras de 32 bits (XLC 17)

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.3.5 Aplicación sin hebras de 64 bits (XLC 17)

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 Aplicación con hebras de 64 bits (XLC 17)

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i Compilación de programas C++ en IBM i

Compile programas en C++ de IBM MQ en IBM i utilizando el compilador ILE C++.

IBM ILE C++ para IBM i es un compilador nativo de programas en C++. Las instrucciones siguientes describen cómo utilizar este compilador para crear aplicaciones C++ de IBM MQ utilizando el programa de ejemplo *Hello World!* IBM MQ como ejemplo.

1. Instale el compilador ILE C++ for IBM i tal como se indica en *Léame primero* manual que acompaña al producto.
2. Asegúrese de que la biblioteca QCXXN esté en la lista de bibliotecas.
3. Cree el programa de ejemplo HELLO WORLD:
   a. Cree un módulo:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   El código fuente en C++ de los programas de ejemplo se puede encontrar en `/QIBM/ProdData/mqm/samp` y los archivos de inclusión en `/QIBM/ProdData/mqm/inc`.

   De forma alternativa, el código fuente se puede encontrar en la biblioteca `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

   b. Enlace esto con programas de servicio proporcionados por IBM MQ para generar un objeto de programa:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
   ```

   Para compilar una aplicación con hebras, la aplicación usa los programas de servicio reentrante:

   ```
   CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
   BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
   ```

   c. Ejecute el programa de ejemplo HELLO WORLD utilizando SYSTEM.DEFAULT.LOCAL.QUEUE:

   ```
   CALL PGM(MYLIB/IMQWRLD)
   ```

Linux Compilación de programas C++ en Linux

Compile programas C++ de IBM MQ en Linux utilizando el compilador GNU g++.

System p

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Cliente: System p

Aplicación sin hebras de 32 bits

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Servidor: System p

Aplicación sin hebras de 32 bits

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
```

```
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Cliente: IBM Z

Aplicación sin hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Servidor: IBM Z

Aplicación sin hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Cliente: x86-64 (32 bits)

Aplicación sin hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Servidor: x86-64 (32 bits)

Aplicación sin hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
```

```
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows Compilación de programas C++ en Windows

Compile programas C++ de IBM MQ en Windows utilizando el compilador C++ de Microsoft Visual Studio.

Atención: Las bibliotecas que entrega IBM MQ son bibliotecas dinámicas y no bibliotecas estáticas. IBM MQ proporciona algo conocido como "`import libraries`" que solo puede utilizar durante el tiempo de compilación. Durante el tiempo de ejecución, debe utilizar las bibliotecas dinámicas.

A partir de IBM MQ 8.0.0 Fix Pack 4, IBM MQ proporciona clientes redistribuibles, que contienen las bibliotecas necesarias para ejecutar aplicaciones IBM MQ. Estas bibliotecas se pueden empaquetar y redistribuir con aplicaciones cliente. Para obtener más información, consulte Clientes redistribuibles en Windows.

Los archivos de biblioteca (.lib) y los archivos dll para su uso con aplicaciones de 32 bits están instalados en *MQ_INSTALLATION_PATH*`/Tools/Lib`. Los archivos a utilizar con aplicaciones de 64 bits están instalados en *MQ_INSTALLATION_PATH*`/Tools/Lib64`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.

Cliente

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

Servidor

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Instalación del entorno de ejecución C universal

Si utiliza Windows 8.1 o Windows Server 2012 R2, debe instalar la actualización del entorno de ejecución C universal (Universal CRT) desde Microsoft. Este entorno de ejecución se incluye como parte de Windows 10 y de Windows Server 2016.

La actualización Universal CRT es la actualización de Microsoft KB3118401. Puede comprobar si tiene esta actualización buscando un archivo denominado `ucrtbase.dll` en el directorio `C:\Windows\System32`. Si no es así, puede descargar la actualización desde la página de Microsoft siguiente: https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

El intento de ejecutar un programa de IBM MQ o un programa que compile usted mismo utilizando Microsoft Visual Studio 2017 sin el entorno de ejecución instalado, provocará errores como el siguiente:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Suministro de entornos de ejecución para programas de Microsoft Visual Studio 2012

Si ha compilado un programa IBM MQ utilizando Microsoft Visual Studio 2012, tenga en cuenta que el instalador de IBM MQ no instala los entornos de ejecución C/C++ de Microsoft Visual Studio 2012. Si la versión anterior de IBM MQ se instaló en el mismo sistema, los entornos de ejecución de Microsoft Visual Studio 2012 estarán disponibles desde dicha instalación.

No obstante, si utiliza un programa que se ha compilado utilizando Microsoft Visual Studio 2012 y no se ha instalado ninguna versión anterior de IBM MQ, deberá realizar una de las acciones siguientes:

- Descargue e instale **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** desde Microsoft.
- Volver a compilar el programa con Microsoft Visual Studio 2017 u otro nivel de Microsoft Visual Studio para el que estén instalados los entornos de ejecución.

Bibliotecas de cliente C++ creadas mediante el compilador de Microsoft Visual Studio 2015

IBM MQ proporciona bibliotecas de cliente de C++ compiladas con el compilador C++ de Microsoft Visual Studio 2015 y el compilador C++ de Microsoft Visual Studio 2017.

Se proporcionan tanto la versión de 32 bits como la versión de 64 bits de las bibliotecas C++ de IBM MQ. Las bibliotecas de 32 bits se instalan en la carpeta `bin\vs2015` y las bibliotecas de 64 bits se instalan en las carpetas `bin64\vs2015`.

De forma predeterminada, IBM MQ está configurado para utilizar las bibliotecas de Microsoft Visual Studio 2017. Para utilizar las bibliotecas de Microsoft Visual Studio 2015, debe establecer la variable de entorno `MQ_PREFIX_VS_LIBRARIES` en `MQ_PREFIX_VS_LIBRARIES=vs2015` antes de instalar IBM MQ, o antes de utilizar los mandatos **setmqenv** o **setmqinst**.

Utilización de bibliotecas C++ de IBM MQ con distinto nombre

IBM MQ proporciona algunas bibliotecas de cliente C++ adicionales que se denominan de forma distinta. Estas bibliotecas están compiladas con los compiladores C++ de Microsoft Visual Studio 2015 y Microsoft Visual Studio 2017. Estas bibliotecas se proporcionan además de las bibliotecas C++ existentes compiladas también con el compilador C++ de Microsoft Visual Studio 2017. Dado que estas bibliotecas C++ de IBM MQ tienen nombres distintos, puede ejecutar aplicaciones C++ de IBM MQ que se hayan creado utilizando C++ de IBM MQ y se hayan compilado con Microsoft Visual Studio 2017 y versiones anteriores del producto en el mismo sistema.

Las bibliotecas adicionales de Microsoft Visual Studio 2017 tienen los nombres siguientes:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

Las bibliotecas adicionales de Microsoft Visual Studio 2015 tienen los nombres siguientes:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Se proporcionan tanto la versión de 32 bits como la versión de 64 bits de estas bibliotecas. Las bibliotecas de 32 bits se instalan en la carpeta `bin` y las bibliotecas de 64 bits se instalan en la carpeta `bin64`. Las bibliotecas de importación correspondientes se instalan en los directorios `Tools\lib` y `Tools\lib64`.

Si su aplicación utiliza archivos `imq*vs2015.lib`, debe compilarlos utilizando el compilador de Microsoft Visual Studio 2015. Para ejecutar aplicaciones C++ de IBM MQ compiladas con Microsoft Visual Studio 2015, o aplicaciones compiladas con una versión anterior del producto en el mismo sistema, debe añadirse un prefijo a la variable PATH, tal como se muestra en los ejemplos siguientes:

- Para aplicaciones de 32 bits:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Para aplicaciones de 64 bits:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

Conceptos relacionados

Windows: Cambios en IBM MQ 8.0

z/OS Creación de programas C++ en z/OS Batch, RRS Batch y CICS

Cree programas IBM MQ C++ en z/OS para los entornos por lotes, RRS o CICS y ejecute los programas de ejemplo.

Puede escribir programas C++ para tres de los entornos soportados por IBM MQ for z/OS:

- Lote
- Lote RRS
- CICS

Compilar, preenlazar y enlazar

Cree una aplicación z/OS compilando, preenlazando y editando enlaces en su código fuente C++.

IBM MQ C++ para z/OS se implementa como las DLL de z/OS para el lenguaje C++ de IBM para z/OS. Mediante estas DLL puede concatenar las unidades laterales de definición suministradas con la salida del compilador durante el preenlace. Esto permite al enlazador comprobar sus llamadas con las funciones de miembros de C++ de IBM MQ.

Nota: Hay tres conjuntos de unidades laterales para cada uno de los tres entornos.

Para crear una aplicación IBM MQ for z/OS C++, cree y ejecute JCL. Siga este procedimiento:

1. Si su aplicación se ejecuta en CICS, utilice el procedimiento que proporciona CICS para convertir los mandatos CICS en su programa.

   Asimismo, para las aplicaciones CICS necesita:

   a. Añadir la biblioteca SCSQLOAD a la concatenación DFHRPL.

   b. Definir el grupo CSQCAT1 CEDA utilizando el miembro IMQ4B100 de la biblioteca SCSQPROC.

   c. Instalar CSQCAT1.
2. Compile el programa para generar el código de objeto. El JCL de su compilación debe incluir sentencias que hagan que los archivos de definición de datos del producto estén disponibles para el compilador. Las definiciones de datos se proporcionan en las siguientes bibliotecas de IBM MQ for z/OS:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Asegúrese de que especifica la opción de compilador `/cxx`.

   Nota: El nombre **thlqual** es el cualificador de alto nivel de la biblioteca de instalación de IBM MQ en z/OS.
3. Preenlace el código de objeto que ha creado en el paso “2” en la página 569, incluyendo las siguientes unidades laterales de definición que se proporcionan en **thlqual**.SCSQDEFS:

   a. imqs23dm e imqb23dm para procesos por lotes

   b. imqs23dr e imqb23dr para Lote RRS

   c. imqs23dc e imqb23dc para CICS

   Estas son las DLL correspondientes.

   a. imqs23im e imqb23im para procesos por lotes

   b. imqs23ir e imqb23ir para Lote RRS

c. imqs23ic e imqb23ic para CICS

4. Edite enlaces del código de objeto que ha creado en el paso “3” en la página 569, para generar un módulo de carga y almacenarlo en la biblioteca de carga de la aplicación.

Para ejecutar programas por lotes o Lote RRS, incluya las bibliotecas **thlqual**.SCSQAUTH y **thlqual**.SCSQLOAD en la concatenación del conjunto de datos STEPLIB o JOBLIB.

Para ejecutar un programa CICS, en primer lugar, solicite al administrador del sistema que lo defina en CICS como un programa y transacción de IBM MQ. A continuación, puede ejecutarlo del modo habitual.

Ejecutar los programas de ejemplo

Los programas se describen en la sección “Programas de ejemplo C++” en la página 546.

Las aplicaciones de ejemplo solo se proporcionan en formato de código fuente. Los archivos son:

Tabla 75. Archivos de programas de ejemplo de z/OS

Ejemplo	Programa fuente (en la biblioteca thlqual.SCSQCPPS)	JCL (en la biblioteca thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

Para ejecutar los ejemplos, debe compilarlos y editar enlaces como con cualquier programa C++. Consulte la sección “Creación de programas C++ en z/OS Batch, RRS Batch y CICS” en la página 569. Utilice el JCL proporcionado para crear y ejecutar un trabajo por lotes. Inicialmente debe personalizar el JCL siguiendo el comentario que se incluye con el mismo.

z/OS Compilación de programas C++ en z/OS UNIX System Services

Cree programas IBM MQ C++ en z/OS UNIX System Services (z/OS UNIX).

Para crear una aplicación bajo el shell de z/OS UNIX , debe proporcionar al compilador acceso a los archivos de inclusión IBM MQ (que se encuentran en `thlqual.SCSQC370` y `hlqual.SCSQHPPS`) y enlazar con dos de las unidades laterales DLL (que se encuentran en `thlqual.SCSQDEFS`). En tiempo de ejecución, la aplicación necesita acceder a los IBM MQ conjuntos de datos `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`y uno de los conjuntos de datos específicos del idioma, como `thlqual.SCSQANLE`[6].

Compilación

1. Copie el ejemplo en el sistema de archivos utilizando el mandato TSO **oput** o utilice FTP. En el resto de este ejemplo se asume que se ha copiado el ejemplo en un directorio llamado `/u/fred/sample` y que se ha nombrado `imqwrld.cpp`.
2. Inicie sesión en el shell de z/OS UNIX y vaya al directorio donde ha colocado el ejemplo.
3. Configure el compilador de C++ para que pueda aceptar los archivos de unidad lateral de DLL y .cpp como entrada:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compile y enlace el programa de ejemplo. El comando siguiente enlaza el programa con las unidades laterales de proceso por lotes; en su lugar, se pueden utilizar las unidades laterales por lotes de RRS.

[6] Puede enlazar con cualquiera de las unidades laterales listadas en " Preenlazar el código de objeto para ejecutar z/OS UNIX en cualquiera de los tres entornos, “Creación de programas C++ en z/OS Batch, RRS Batch y CICS” en la página 569

El carácter \ se utiliza para dividir el comando en más de una línea. No introduzca este carácter; ejecute el comando en una única línea:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

Para obtener más información sobre el mandato TSO **oput** , consulte la publicación z/OS UNIX Command Reference.

También se puede usar la utilidad make para simplificar la creación de programas C++. A continuación se muestra un ejemplo de archivo makefile para compilar el programa de ejemplo HELLO WORLD en C++. Separa las fases de compilación y enlazado. Configure el entorno como en el paso “3” en la página 570 antes de ejecutar make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Consulte z/OS UNIX System Services Programming Tools para obtener más información sobre cómo utilizar make.

En ejecución

1. Inicie sesión en el shell de z/OS UNIX y vaya al directorio en el que ha creado el ejemplo.
2. Configure la variable de entorno STEPLIB para incluir los conjuntos de datos IBM MQ:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. Ejecute el ejemplo:

```
/u/fred/sample:> ./imqwrld
```

Desarrollo de aplicaciones .NET

IBM MQ classes for .NET permitir que las aplicaciones .NET se conecten a IBM MQ como un IBM MQ MQI client o que se conecten directamente a un servidor IBM MQ .

Si tiene aplicaciones que utilizan Microsoft .NET Framework y desea beneficiarse de los recursos de IBM MQ, debe utilizar IBM MQ classes for .NET. Para obtener más información, consulte “Instalación del IBM MQ classes for .NET Framework” en la página 579.

A partir de IBM MQ 9.1.1, IBM MQ da soporte a .NET Core para aplicaciones en entornos Windows. Para obtener más información, consulte “Instalación del IBM MQ classes for .NET” en la página 572.

A partir de IBM MQ 9.1.2, IBM MQ da soporte a .NET Core para aplicaciones en entornos Linux.

A partir de IBM MQ 9.1.4, las aplicaciones gestionadas de IBM MQ .NET pueden equilibrar automáticamente las conexiones entre los gestores de colas agrupados en clústeres. Se da soporte a las bibliotecas .NET Framework y .NET Standard. Para obtener más información, consulte Acerca de los clústeres uniformes y Equilibrio automático de aplicaciones.

La interfaz de IBM MQ .NET orientada a objetos es distinta de la interfaz de MQI en tanto que utiliza métodos de objetos en lugar de utilizar los verbos de MQI.

La interfaz de programación de aplicaciones de IBM MQ orientada a procedimientos está basada en verbos, tales como los contenidos en la lista siguiente:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Estos verbos toman todos como parámetro un descriptor de contexto del objeto de IBM MQ sobre el que tienen que trabajar. Debido a que .NET está orientado a objetos, la interfaz de programación de .NET invierte esta funcionalidad. El programa del usuario consta de un conjunto de objetos de IBM MQ, sobre los que el usuario actúa invocando métodos sobre esos objetos. Puede escribir programas en cualquier lenguaje soportado por .NET.

Cuando utiliza la interfaz orientada a procedimientos, se desconecta de un gestor de colas mediante la llamada MQDISC(*Hconn*,CompCode, Reason), donde *Hconn* es un descriptor de contexto del gestor de colas.

En la interfaz .NET, el gestor de colas está representado por un objeto de la clase MQQueueManager. Se puede desconectar del gestor de colas invocando el método Disconnect() para esa clase.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

IBM MQ classes for .NET es un conjunto de clases que permiten que las aplicaciones .NET interaccionen con IBM MQ. Esas clases representan los diversos componentes de IBM MQ que son utilizados por la aplicación, tales como gestores de colas, colas, canales y mensajes. Para obtener más detalles de estas clases, consulte Las clases e interfaces de IBM MQ .NET.

Para poder compilar las aplicaciones que escribe, debe tener instalado .NET Framework. Para obtener instrucciones sobre la instalación de IBM MQ classes for .NET y .NET Framework, consulte “Instalación del IBM MQ classes for .NET Framework” en la página 579.

Conceptos relacionados

Visión general técnica

“Desarrollo de aplicaciones para IBM MQ” en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

Tareas relacionadas

Cómo ponerse en contacto con el servicio de soporte de IBM

Resolución de problemas de IBM MQ .NET

“Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ” en la página 1297
El canal personalizado de Microsoft Windows Communication Foundation (WCF) para IBM MQ envía y recibe mensajes entre clientes y servicios WCF.

“Desarrollo de aplicaciones XMS .NET” en la página 633
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) proporciona una interfaz de programación de aplicaciones (API) llamada XMS que tiene el mismo conjunto de interfaces que la API Java Message Service (JMS). IBM MQ Message Service Client (XMS) for .NET contiene una implementación totalmente gestionada de XMS, que puede ser utilizada por cualquier lenguaje compatible con .NET.

Windows Linux Instalación del IBM MQ classes for .NET

IBM MQ classes for .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux

Requisitos previos e instalación

A partir de IBM MQ 9.2.0, la biblioteca de cliente de IBM MQ .NET creada utilizando .NET Standard está disponible en Windows y Linux. Para ejecutar IBM MQ classes for .NET Standard, debe instalar Microsoft .NET Core. Microsoft.NET Core 3.1 es la versión mínima necesaria para ejecutar IBM MQ classes for .NET Standard.

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, IBM MQ da soporte a aplicaciones .NET 6 utilizando IBM MQ classes for .NET Standard. Si está utilizando una aplicación .NET Core 3.1 , puede ejecutar esta aplicación con una edición pequeña en el archivo `csproj` , estableciendo `targetframeworkversion` en `"net6.0"`, sin que sea necesaria ninguna recompilación.

V 9.3.1 IBM MQ 9.3.1 proporciona una biblioteca de cliente de IBM MQ .NET creada en .NET 6 como infraestructura de destino. A partir de IBM MQ 9.3.1, Microsoft .NET 6.0 es la versión mínima necesaria para ejecutar aplicaciones utilizando bibliotecas de IBM MQ que se compilan utilizando .NET 6 como infraestructura de destino.

V 9.3.1 A partir de IBM MQ 9.3.1, la biblioteca de cliente de IBM MQ .NET creada utilizando .NET Standard está disponible en una carpeta nueva `netstandard2.0` y la biblioteca de cliente de IBM MQ .NET creada utilizando .NET 6 como infraestructura de destino está disponible en *MQ_INSTALLATION_PATH*`/bin` en Windows y en *MQ_INSTALLATION_PATH*`/lib64` en Linux.

La versión más reciente de IBM MQ classes for .NET se instala de forma predeterminada como parte de la instalación estándar de IBM MQ en la característica *Java y .NET Messaging and Web Services* .

Windows Si desea más información sobre los requisitos previos y la instalación en Windows:

- Consulte Requisitos para IBM MQ classes for .NET, para que el software de requisito previo ejecute IBM MQ classes for .NET.
- Consulte Instalación del servidor IBM MQ en Windows o Instalación de un cliente IBM MQ en sistemas Windows para ver las instrucciones de instalación .

Linux Si desea más información sobre los requisitos previos y la instalación en Linux:

- Consulte Requisitos para IBM MQ classes for .NET, para que el software de requisito previo ejecute IBM MQ classes for .NET.
- Para ver las instrucciones de instalación de rpm, consulte Instalación de un cliente IBM MQ en sistemas Linux.
- Para Linux Ubuntu, utilizando paquetes Debian, consulte Instalación de un cliente IBM MQ en sistemas Linux.

La biblioteca de IBM MQ classes for .NET Standard , `amqmdnetstd.dll`, está disponible para su descarga desde el repositorio de NuGet . Para obtener más información, consulte “Descarga de IBM MQ classes for .NET desde el repositorio de NuGet” en la página 578.

Biblioteca `amqmdnetstd.dll`

V 9.3.1 A partir de IBM MQ 9.3.1, la biblioteca `amqmdnetstd.dll` está disponible en las ubicaciones siguientes

Biblioteca creada utilizando .NET Standard 2.0 como infraestructura de destino

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`.
- Linux En Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`.

Deprecated Estas bibliotecas están en desuso y IBM tiene previsto eliminarlas en futuros releases.

Biblioteca creada utilizando .NET 6 como infraestructura de destino

- Windows En Windows: *MQ_INSTALLATION_PATH*\bin. Las aplicaciones de ejemplo se instalan en *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.
- Linux En Linux: *MQ_INSTALLATION_PATH*\lib64. Los ejemplos de .NET se encuentran en *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.

LTS Para IBM MQ 9.3.0 Long Term Support, la biblioteca amqmdnetstd.dll está disponible en las ubicaciones siguientes:

- Windows En Windows: *MQ_INSTALLATION_PATH*\bin. Las aplicaciones de ejemplo se instalan en *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.
- Linux En Linux: *MQ_INSTALLATION_PATH*/lib64 path. Los ejemplos de .NET se encuentran en *MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base.

Atención: V 9.3.1 Deprecated A partir de IBM MQ 9.3.1, las bibliotecas de cliente de IBM MQ .NET compiladas utilizando .NET Standard 2.0 como infraestructura de destino están en desuso y las aplicaciones que hacen referencia a estas bibliotecas emiten un aviso CS0618 durante el tiempo de compilación.

LTS Stabilized La biblioteca amqmdnet.dll para .NET Framework se sigue proporcionando, pero esta biblioteca se ha estabilizado; es decir, no se introducirán nuevas características. Para cualquiera de las características más recientes, debe migrar a la biblioteca amqmdnetstd.dll. Sin embargo, puede seguir utilizando la biblioteca amqmdnet.dll en IBM MQ 9.1 o releases posteriores de Long Term Support o Continuous Delivery .

V 9.3.1 Si una aplicación .NET Framework se compila utilizando amqmdnetstd.dll o amqmxmsstd.dll desde una versión inferior a IBM MQ 9.3.1 y la misma aplicación se ejecuta utilizando las bibliotecas de cliente IBM MQ basadas en .NET 6 , .NETemite el siguiente tipo de excepción FileLoadException :

```
Se ha capturado una excepción: System.IO.FileLoadException: No se ha podido cargar el archivo o
ensamblaje
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' o
una de sus dependencias. La definición de manifiesto del ensamblaje ubicado no coincide con
referencia de ensamblaje. (Excepción de HRESULT: 0x80131040)

Nombre de archivo: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Para resolver este error, las bibliotecas presentes en *MQ_INSTALLATION_PATH*/bin/netstandard2.0 deben copiarse en el directorio desde donde se ejecuta la aplicación .NET Framework .

Mandato dspmqver

Puede utilizar el mandato **dspmqver** para visualizar información de versión y compilación para el componente .NET Core .

Comparación de características entre IBM MQ classes for .NET Framework y IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6)

En la tabla siguiente se listan las características de IBM MQ classes for .NET Framework en comparación con las características de IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6).

Tabla 76. Diferencias entre IBM MQ classes for .NET Framework y IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6)

Característica	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6)**
Nombres de clase (API)	Todas las clases siguen siendo las mismas en cada red.	Todas las clases siguen siendo las mismas en cada red.
Sistema operativo	Windows	Windows Contenedores con Docker Linux macOS
Archivo `app.config` (archivo de configuración para habilitar el rastreo en el cliente redistribuible)	El archivo `app.config` se utiliza para habilitar el rastreo para el paquete redistribuible y el cliente autónomo de IBM MQ .NET . Consulte Rastreo de un cliente IBM MQ classes for .NET Framework utilizando un archivo de configuración de aplicación para obtener más información sobre las variables que utiliza para el rastreo, incluidos **MQTRACEPATH** y **MQTRACELEVEL**.	`app.config` No recibe soporte. Utilice variables de entorno.

Tabla 76. Diferencias entre IBM MQ classes for .NET Framework y IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6) (continuación)

Característica	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6)
Rastreo	Para una instalación de cliente completa de IBM MQ, puede utilizar el mandato **strmqtrc** para habilitar el rastreo para IBM MQ classes for .NET Framework. Para los clientes redistribuibles, el archivo `app.config` también se utiliza para habilitar el rastreo. Para obtener más información, consulte Rastreo de aplicaciones de IBM MQ .NET. V 9.3.3 A partir de IBM MQ 9.3.3, puede habilitar e inhabilitar el rastreo utilizando el archivo `mqclient.ini` y estableciendo las propiedades adecuadas de la stanza Trace. También puede habilitar e inhabilitar dinámicamente el rastreo con el archivo `mqclient.ini` . Para obtener más información, consulte Rastreo de aplicaciones de IBM MQ .NET con mqclient.ini.	La variable de entorno **MQDOTNET_TRACE_ON** se utiliza para habilitar el rastreo para clientes redistribuibles. Los valores iguales e inferiores a 0 no habilitan el rastreo. Un valor de 1 habilita el rastreo de nivel predeterminado. Un valor mayor que 1, habilita el rastreo detallado. Si se establece esta variable de entorno en cualquier otro valor como, por ejemplo, string, no se habilita el rastreo. Consulte Rastreo de aplicaciones IBM MQ .NET utilizando variables de entorno. La variable de entorno **MQDOTNET_TRACE_ON** comprueba si el directorio de rastreo IBM MQ está disponible o no. Si el directorio de rastreo está disponible, el archivo de rastreo se genera en el directorio de rastreo. Sin embargo, si IBM MQ no está instalado, el archivo de rastreo se copia en el directorio de trabajo actual. Otras variables de entorno, incluyendo **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**, etc., que se utilizan para IBM MQ classes for .NET Framework, se pueden utilizar y trabajar de la misma forma. V 9.3.3 A partir de IBM MQ 9.3.3, puede habilitar e inhabilitar el rastreo utilizando el archivo `mqclient.ini` y estableciendo las propiedades adecuadas de la stanza Trace. También puede habilitar e inhabilitar dinámicamente el rastreo con el archivo `mqclient.ini` . Para obtener más información, consulte Rastreo de aplicaciones de IBM MQ .NET con mqclient.ini.
Modalidades de transporte	Gestionado, no gestionado y enlaces	Gestionado

Tabla 76. Diferencias entre IBM MQ classes for .NET Framework y IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6) (continuación)

Característica	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6)
TLS	El almacén de claves Windows se utiliza para almacenar los certificados.	Windows En Windows, se debe utilizar el almacén de claves para almacenar los certificados. Los valores permitidos son *USER o *SYSTEM. Basándose en la entrada, el cliente de IBM MQ .NET busca en el almacén de claves de Windows del usuario actual, o en todo el sistema. Linux En Linux, se recomienda utilizar la clase X509Store para instalar certificados y .NET Core instala certificados en la siguiente ubicación: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Soportado	Soportado, y los valores de la vía de acceso CCDT son los mismos que para las clases de .NET Framework.
Reconexión automática de cliente	Soportado	Soportado
Transacciones distribuidas	Soportado	No soportado
Instalación de bibliotecas de enlace dinámico (archivos dll) en la memoria caché del conjunto global (GAC)	Los archivos Dll se instalan en la GAC como parte de la instalación de IBM MQ.	Los archivos de Dll no se instalan en la GAC como parte de la instalación de IBM MQ.

Nota: Windows identificadores de seguridad (SID) Windows:

La autenticación a nivel de dominio no está soportada para IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6). El ID de usuario que ha iniciado sesión se utiliza para la autenticación.

Desarrollo de aplicaciones IBM MQ .NET Core en macOS

macOS

Las aplicaciones de IBM MQ .NET Core se pueden desarrollar en macOS.

Las bibliotecas IBM MQ .NET no se empaquetan con el kit de herramientas de macOS, de forma que debe copiarlas desde un cliente Windows o Linux IBM MQ en macOS. A continuación, puede utilizar estas bibliotecas para desarrollar aplicaciones IBM MQ .NET Core en macOS.

Una vez desarrolladas, estas aplicaciones se pueden ejecutar con soporte en entornos Windows o Linux.

Conceptos relacionados

"Instalación del IBM MQ classes for .NET Framework" en la página 579
IBM MQ classes for .NET Framework, incluidos los ejemplos, se instalan con IBM MQ. Se trata de un requisito previo de Microsoft.NET Framework en Windows.

"Instalación del IBM MQ classes for XMS .NET" en la página 637

IBM MQ classes for XMS .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux.

Windows Linux Descarga de IBM MQ classes for .NET desde el repositorio de NuGet

Los IBM MQ classes for .NET están disponibles para su descarga desde el repositorio de NuGet , para que los desarrolladores de .NET puedan consumirlos fácilmente.

Acerca de esta tarea

NuGet es el gestor de paquetes para las plataformas de desarrollo de Microsoft, incluyendo .NET. Las herramientas de cliente de NuGet proporcionan la capacidad de producir y consumir paquetes. Un paquete NuGet es un único archivo comprimido con la extensión `.nupkg` que contiene código compilado (DLL), otros archivos relacionados con ese código y un manifiesto descriptivo que incluye información como el número de versión del paquete.

Puede descargar el paquete de `IBMMQDotnetClient` NuGet , que contiene la biblioteca `amqmdnetstd.dll` , desde NuGet Gallery, que es el repositorio central de paquetes utilizado por todos los autores y consumidores de paquetes.

Nota: V 9.3.1 A partir de IBM MQ 9.3.1, el paquete NuGet contiene bibliotecas compiladas utilizando .NET Standard 2.0 y .NET 6 como infraestructura de destino. Las bibliotecas para .NET Standard 2.0 están disponibles en la carpeta `netstandard2.0` y las bibliotecas para .NET 6 están disponibles en la carpeta `net6.0` .

Existen tres formas de descarga del paquete `IBMMQDotnetClient`:

- Utilizando Microsoft Visual Studio. NuGet se distribuye como una extensión de Microsoft Visual Studio. Desde Microsoft Visual Studio 2012, NuGet se instala previamente de forma predeterminada.
- Desde la línea de mandatos, utilizando el gestor de paquetes NuGet, o bien la CLI de .NET.
- Utilizando un navegador web.

En cuanto al paquete redistribuible, habilite el rastreo utilizando la variable de entorno **MQDOTNET_TRACE_ON**.

Procedimiento

- Para descargar el paquete `IBMMQDotnetClient` utilizando la interfaz de usuario del gestor de paquetes en Microsoft Visual Studio, complete los pasos siguientes:
  a) Pulse con el botón derecho del ratón en el proyecto .NET y, a continuación, pulse **Gestionar paquetes Nuget**.
  b) Pulse la pestaña **Examinar** y busque `"IBMMQDotnetClient"`.
  c) Seleccione el paquete y pulse **Instalar**.

  Durante la instalación, el gestor de paquetes proporciona información de progreso en forma de sentencias de consola.
- Para descargar el paquete `IBMMQDotnetClient` desde la línea de mandatos, elija una de las opciones siguientes:
  - Utilizando el gestor de paquetes NuGet, entre el mandato siguiente:

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    Durante la instalación, el gestor de paquetes proporciona información de progreso en forma de sentencias de consola. Puede redirigir la salida a un archivo de registro.
  - Utilizando la CLI de .NET, especifique el mandato siguiente:

    ```
    dotnet add package IBMMQDotnetClient --version 9.1.4
    ```

- Utilizando un navegador web, descargue el paquete `IBMMQDotnetClient` desde https://www.nuget.org/packages/IBMMQDotnetClient.

Conceptos relacionados

Información de licencia del Cliente de IBM MQ para .NET

Tareas relacionadas

"Descarga de IBM MQ classes for XMS .NET desde el repositorio NuGet" en la página 640

Los IBM MQ classes for XMS .NET están disponibles para su descarga desde el repositorio de NuGet , para que los desarrolladores de .NET puedan consumirlos fácilmente.

Windows Instalación del IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework, incluidos los ejemplos, se instalan con IBM MQ. Se trata de un requisito previo de Microsoft.NET Framework en Windows.

La última versión de IBM MQ classes for .NET Framework se instala de forma predeterminada como parte de la instalación estándar de IBM MQ en la función *Mensajería de Java y .NET y servicios web*. Para obtener instrucciones de instalación, consulte Instalación del servidor IBM MQ en Windows o Instalación de un cliente de IBM MQ en sistemas Windows.

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, para ejecutar IBM MQ classes for .NET Framework , debe instalar Microsoft.NET Framework V4.7.2 o posterior. Se trata de un cambio de IBM MQ 9.2 donde la versión mínima necesaria era V4.6.2.

V 9.3.0 V 9.3.0 Las aplicaciones existentes que se compilan con Microsoft.NET Framework V3.5 se pueden ejecutar sin volver a compilar añadiendo el código siguiente al archivo `app.config` de la aplicación:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

Nota: Si Microsoft .NET Framework V4.7.2 o superior no está instalado antes de instalar IBM MQ, la instalación del producto IBM MQ continúa sin errores, pero IBM MQ classes for .NET no está disponible. Si .NET Framework se instala después de instalar IBM MQ, los ensamblajes de IBM MQ.NET se deben registrar ejecutando el script *`WMQInstallDir`*`\bin\amqiRegisterdotNet.cmd`, donde *WMQInstallDir* es el directorio donde está instalado IBM MQ. Este script instala los ensamblajes necesarios en la memoria caché de ensamblaje global (GAC). Un conjunto de archivos `amqi*.log` que registran las acciones que se realizan se crean en el directorio %TEMP% . No es necesario volver a ejecutar el script `amqiRegisterdotNet.cmd` si .NET se actualiza a V4.7.2 o superior desde una versión anterior, por ejemplo, desde .NET V3.5.

En un entorno de instalación múltiple, si ha instalado previamente IBM MQ classes for .NET como paquete de soporte, no puede instalar IBM MQ a menos que primero desinstale el paquete de soporte. El producto IBM MQ classes for .NET que se instala con IBM MQ contiene las mismas funciones que el paquete de soporte.

También se proporcionan aplicaciones de ejemplo, así como archivos fuente. Consulte "Aplicaciones de ejemplo para .NET" en la página 580.

Para obtener más información sobre cómo utilizar el canal personalizado de IBM MQ para Microsoft WCF con .NET, consulte "Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ" en la página 1297

Conceptos relacionados

"Instalación del IBM MQ classes for .NET" en la página 572

IBM MQ classes for .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux

Tareas relacionadas

Rastreo de aplicaciones .NET de IBM MQ

Opciones para conectar IBM MQ classes for .NET a un gestor de colas

Hay tres modalidades para conectar IBM MQ classes for .NET a un gestor de colas. Estudie qué tipo de conexión se ajusta mejor a sus requisitos.

Conexión de enlaces de cliente

Para utilizar IBM MQ classes for .NET como un IBM MQ MQI client, puede instalarlo, con el IBM MQ MQI client, en la máquina del servidor IBM MQ o en una máquina aparte. Los enlaces de cliente pueden usar transacciones XA u otras que no lo sean

Conexión de enlaces de servidor

Cuando se utiliza en la modalidad de enlaces de servidor, IBM MQ classes for .NET utilizan la API del gestor de colas, en lugar de comunicarse a través de una red. Esto proporciona un mejor rendimiento para las aplicaciones de IBM MQ que el uso de conexiones de red.

Para utilizar la conexión de enlaces, debe instalar IBM MQ classes for .NET en el servidor de IBM MQ.

Conexión de cliente gestionado

Una conexión realizada en este modo se conecta como un cliente de IBM MQ a un servidor de IBM MQ que se ejecuta en la máquina local o en una remota.

El IBM MQ classes for .NET que se conecta en esta modalidad permanece en el código gestionado de .NET y no hace llamadas a los servicios nativos. Para obtener más información sobre código gestionado, consulte la documentación de Microsoft.

Hay una serie de limitaciones para utilizar el cliente gestionado. Para obtener más información sobre estos, consulte “Conexiones de cliente gestionado” en la página 596.

Aplicaciones de ejemplo para .NET

Para ejecutar sus propias aplicaciones .NET, utilice las instrucciones para los programas de verificación, sustituyendo el nombre de su aplicación por el de las aplicaciones de ejemplo.

Se suministran las aplicaciones de ejemplo siguientes:

- Una aplicación de colocación de mensajes
- Una aplicación de obtención de mensajes
- Una aplicación 'hello world'
- Una aplicación de publicación/suscripción
- Una aplicación que utiliza propiedades de mensajes

Todas estas aplicaciones se proporcionan en el lenguaje C# y algunas también se proporcionan en C++ y Visual Basic. Puede escribir aplicaciones en cualquier lenguaje soportado por .NET.

Programa SPUT de "Transferencia de mensaje" (nmqsput.cs, mmqsput.cpp, vmqsput.vb)
Este programa muestra cómo colocar un mensaje en una cola concreta. El programa tiene tres parámetros:

- El nombre de una cola (necesario), por ejemplo, SYSTEM.DEFAULT.LOCAL.QUEUE
- El nombre de un gestor de colas (opcional)
- La definición de un canal (opcional), por ejemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si no se especifica ningún gestor de colas, se utiliza el gestor de colas local predeterminado. Si se define un canal, éste tiene el mismo formato que la variable de entorno MQSERVER.

Programa SGET de "Obtención de mensaje" (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Este programa muestra cómo obtener un mensaje en una cola concreta. El programa tiene tres parámetros:

- El nombre de una cola (necesario), por ejemplo, SYSTEM.DEFAULT.LOCAL.QUEUE
- El nombre de un gestor de colas (opcional)
- La definición de un canal (opcional), por ejemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si no se especifica ningún gestor de colas, se utiliza el gestor de colas local predeterminado. Si se define un canal, éste tiene el mismo formato que la variable de entorno MQSERVER.

Programa "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
Este programa muestra cómo colocar y obtener un mensaje de una cola concreta. El programa tiene tres parámetros:

- El nombre de una cola (opcional), por ejemplo, SYSTEM.DEFAULT.LOCAL.QUEUE o SYSTEM.DEFAULT.MODEL.QUEUE
- El nombre de un gestor de colas (opcional)
- Una definición de canal (opcional), por ejemplo, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si no se proporciona un nombre de cola, de forma predeterminada el nombre es SYSTEM.DEFAULT.LOCAL.QUEUE. Si no se especifica ningún gestor de colas, se utiliza el gestor de colas local predeterminado.

Programa de "Publicación/suscripción" (MQPubSubSample.cs)
Este programa muestra cómo utilizar la publicación/suscripción de IBM MQ. Solo se proporciona en C#. El programa tiene dos parámetros:

- El nombre de un gestor de colas (opcional)
- Una definición de canal (opcional)

Programa "Propiedades de mensaje" (MQMessagePropertiesSample.cs)
Este programa muestra cómo utilizar las propiedades del mensaje. Solo se proporciona en C#. El programa tiene dos parámetros:

- El nombre de un gestor de colas (opcional)
- Una definición de canal (opcional)

Puede verificar la instalación compilando y ejecutando estas aplicaciones.

Ubicaciones de instalación

Las aplicaciones de ejemplo se instalan en las ubicaciones siguientes, según el lenguaje en el que se escriben. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.

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

Creación de las aplicaciones de ejemplo

Para crear las aplicaciones de ejemplo, se proporciona un archivo de proceso por lotes para cada idioma.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

El archivo bldcssamp.bat contiene una línea para cada ejemplo, que es todo lo necesario para compilar este programa de ejemplo:

```
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

El archivo bldmcpsamp.bat contiene una línea para cada ejemplo, que es todo lo necesario para compilar este programa de ejemplo:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Si desea compilar estas aplicaciones en Microsoft Visual Studio 2003/.NET SDKv1.1, sustituya el mandato compile:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

por

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

El archivo bldvbsamp.bat contiene una línea para cada ejemplo, que es todo lo necesario para compilar este programa de ejemplo:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Ejemplos para utilizar IBM MQ con Microsoft .NET Core

A partir de IBM MQ 9.2.0, IBM MQ da soporte a .NET Core para aplicaciones IBM MQ .NET en entornos Windows . IBM MQ classes for .NET Standard, incluidos los ejemplos, se instalan de forma predeterminada como parte de la instalación estándar de IBM MQ.

Las aplicaciones de ejemplo para IBM MQ .NET se instalan en `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. También se proporciona un script, que se puede utilizar para compilar los ejemplos.

Puede crear los ejemplos utilizando los archivos `build.bat` proporcionados. Existe un `build.bat` para cada ejemplo en la ubicación siguiente en Windows:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux A partir de IBM MQ 9.2.0, IBM MQ también da soporte a Core para aplicaciones en entornos Linux .

Para obtener más información sobre cómo utilizar IBM MQ con Microsoft .NET Core, consulte “Instalación del IBM MQ classes for .NET” en la página 572.

Configuración del gestor de colas para aceptar conexiones de cliente TCP/IP

Configure un gestor de colas para aceptar solicitudes de conexión entrantes procedentes de los clientes.

Acerca de esta tarea

Esta tarea explica los pasos básicos para configurar un gestor de colas para aceptar conexiones de cliente TCP/IP. En el caso de un sistema de producción, también debe tener en cuenta consideraciones sobre seguridad al configurar gestores de colas.

Procedimiento

1. Defina un canal de conexión de servidor:
   a. Inicie el gestor de colas.
   b. Defina un canal de ejemplo denominado NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   Importante: Este ejemplo está pensado para su uso solamente en un entorno aislado, pues no incluye ninguna consideración sobre seguridad. En el caso de un sistema de producción, considere la posibilidad de utilizar TLS o una salida de seguridad. Consulte Protección IBM MQ para obtener más información.
2. Inicie un proceso de escucha:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   Nota: Los corchetes indican parámetros opcionales; *qmname* no es necesario para el gestor de colas predeterminado, y el número de puerto *número_puerto* no es necesario si utiliza el valor predeterminado (1414).

Transacciones distribuidas en .NET

Las transacciones distribuidas o globales permiten a las aplicaciones clientes incluir en una transacción diversos recursos de datos en dos o más sistemas en red.

En las transacciones distribuidas, un gestor de transacciones coordina y gestiona la transacción entre dos o más gestores de recursos.

Las transacciones pueden ser un proceso de confirmación de fase única o de dos fases. La confirmación en una sola fase es un proceso en el que solo participa un gestor de recursos en el proceso de transacción y en un proceso de confirmación en dos fases hay más de un gestor de recursos participando en la transacción. En el proceso de confirmación en dos fases, el gestor de transacciones envía una llamada de preparación para comprobar si todos los gestores de recursos están preparados para confirmar. Cuando se recibe el acuse de recibo de todos los gestores de recursos, se emite la llamada de confirmación. En caso contrario, tendrá lugar una retrotracción de toda la transacción. Consulte Gestión y soporte de transacciones para obtener más detalles. Los gestores de recursos tienen que informar a los gestores de transacciones de su participación en la transacción. Cuando el gestor de recursos informa al gestor

de transacciones de su participación, el gestor de recursos obtiene llamadas de retorno (callbacks) procedentes del gestor de transacciones cuando la transacción se va a confirmar o retrotraer.

Las clases de IBM MQ .NET ya soportan las transacciones distribuidas en las conexiones de modalidad no gestionada y de enlaces de servidor. En estas modalidades, las clases IBM MQ .NET delegan todas sus llamadas al cliente de transacciones ampliado en C, que gestiona el proceso de transacciones en nombre de .NET.

Ahora las clases IBM MQ.NET dan soporte a las transacciones distribuidas en la modalidad gestionada donde las clases IBM MQ .NET utilizan el espacio de nombres System.Transactions para el soporte de transacciones distribuidas. La infraestructura System.Transactions hace que la programación transaccional sea fácil y eficiente al soportar las transacciones iniciadas en todos los gestores de recursos, incluido IBM MQ. La aplicación IBM MQ .NET puede colocar y obtener mensajes utilizando la programación de transacciones implícitas .NET o el modelo de programación de transacciones explícitas. En las transacciones implícitas, los límites de la transacción los crea el programa de aplicación que decide cuándo se confirma, retrotrae (en el caso de las transacciones explícitas) o completa la transacción. En las transacciones explícitas, hay que especificar explícitamente si se desea confirmar, retrotraer y completar la transacción.

IBM MQ.NET utiliza el coordinador de transacciones distribuidas Microsoft (MS DTC) como gestor de transacciones, que coordina y gestiona la transacción entre varios gestores de recursos. IBM MQ se utiliza como gestor de recursos. Tenga en cuenta que no se puede utilizar TLS con transacciones XA. Hay que usar CCDT. Para obtener más información, consulte Utilización del cliente transaccional extendido con canales TLS.

IBM MQ.NET sigue el modelo de procesamiento de transacciones distribuidas (DTP) X/Open. El modelo de procesamiento de transacciones distribuidas de X/Open es un modelo de procesamiento de transacciones distribuidas propuesto por Open Group, un consorcio de proveedores. Este modelo es un estándar entre la mayoría de los proveedores comerciales en los ámbitos del procesamiento de transacciones y de bases de datos. La mayoría de los productos de gestión de transacciones comerciales soportan el modelo X/DTP.

Modos de transacción

- “Transacciones distribuidas en la modalidad gestionada de .NET” en la página 585
- Transacciones distribuidas para la modalidad no gestionada

Coordinación de transacciones en diversos escenarios

- Una conexión puede participar en varias transacciones, pero solo una de ellas estará activa en un momento dado.
- Durante una transacción, se respeta la llamada MQQueueManager.Disconnect. En este caso, se pide a la transacción que se retrotraiga.
- Durante una transacción, se respetan las llamadas MQQueue.Close o MQTopic.Close. En este caso, se pide a la transacción que se retrotraiga.
- Los límites de transacción los crea el programa de aplicación que decide cuándo se confirma, retrotrae (en el caso de las transacciones explícitas) o completa (en el caso de las implícitas) la transacción.
- Si la aplicación cliente se interrumpe durante una transacción con un error inesperado antes de emitir una llamada Put o Get en una cola o tema, la transacción se retrotrae y se lanza una MQException.
- Si se devuelve el código de razón MQCC_FAILED durante una llamada Put o Get en una cola o tema, se emite una MQException con código de razón y la transacción se retrotrae. Si el gestor de transacciones ya ha emitido una llamada de preparación, IBM MQ .NET devuelve la solicitud de preparación al retrotraer la transacción por la fuerza. Luego el gestor de transacción DTC provoca una retrotracción del trabajo actual en todos los gestores de recursos en las transacciones ambientales actuales.
- Durante una transacción que implica que implique varios gestores de recursos, si por alguna razón de entorno la llamada Put o Get se cuelga indefinidamente, el gestor de transacciones esperará un tiempo estipulado. Una vez vencido el plazo, provoca la retrotracción de todo el trabajo actual en todos los

gestores de recursos en las transacciones ambientales actuales. Si esta espera indefinida se produce durante la fase de preparación, podría agotarse el tiempo de espera del gestor de transacciones o emitirse una llamada en duda en el recurso, en cuyo caso se retrotraería la transacción.

- Las aplicaciones que utilizan transacciones tienen que colocar (Put) u obtener (Get) mensajes bajo SYNC_POINT. Si se emite una llamada Put o Get de un mensaje bajo un contexto transaccional que no esté bajo SYNC_POINT, la llamada fallará con el código de razón MQRC_UNIT_OF_WORK_NOT_STARTED.

Diferencias de comportamiento entre el soporte de transacciones de cliente gestionado y no gestionado utilizando el espacio de nombres System.Transactions de Microsoft.NET

Las transacciones anidadas tienen un TransactionScope dentro de otro TransactionScope

- El cliente IBM MQ .NET totalmente gestionado da soporte al TransactionScope anidado
- El cliente IBM MQ .NET no gestionado no da soporte al TransactionScope anidado

Transacciones dependientes en System.Transactions

- El cliente IBM MQ .NET totalmente gestionado da soporte al recurso de transacciones dependiente proporcionado por System.Transactions.
- El cliente IBM MQ .NET no gestionado no da soporte al recurso de transacciones dependiente proporcionado por System.Transactions.

Ejemplos de producto

Los ejemplos de producto `SimpleXAPuty` `SimpleXAGet` están disponibles en `WebSphere MQ\tools\dotnet\samples\cs\base`. Los ejemplos son aplicaciones C#, que ejemplifican el uso de MQPUT y MQGET en transacciones distribuidas utilizando el espacio de nombres SystemTransactions. Para obtener más información sobre estos ejemplos, consulte “Creación de mensajes de colocación y obtención sencillos dentro de un TransactionScope (ámbito transaccional)” en la página 588.

Transacciones distribuidas en la modalidad gestionada de .NET

Las clases de IBM MQ .NET utilizan el espacio de nombres System.Transactions para el soporte de transacciones distribuidas en modalidad gestionada. En la modalidad gestionada, MS DTC coordina y gestiona las transacciones distribuidas en todos los servidores incluidos en una transacción.

Las clases de IBM MQ .NET proporcionan un modelo de programación explícito basado en la clase System.Transactions.Transaction y un modelo de programación implícito utilizando la clase System.Transactions.TransactionScope, donde la infraestructura gestiona automáticamente las transacciones.

Transacción implícita

El fragmento de código siguiente describe cómo una aplicación de IBM MQ .NET coloca un mensaje utilizando la programación de transacciones implícita de .NET.

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Explicación del flujo de código de la transacción implícita

El código crea *TransactionScope* y coloca el mensaje bajo el ámbito. A continuación, invoca *Complete* para informar al coordinador de transacciones de la finalización de la transacción. El coordinador de transacciones emite *prepare* y *commit* para completar la transacción. Si se detecta un problema, se invoca *rollback*.

Transacción explícita

El siguiente código describe cómo una aplicación de IBM MQ .NET coloca mensajes utilizando el modelo de programación de transacciones explícito de .NET.

```
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();
}
```

Explicación del flujo de código de la transacción explícita

El fragmento de código crea la transacción utilizando la clase *CommitableTransaction*. Coloca un mensaje debajo de ese ámbito y, a continuación, invoca explícitamente *commit* para completar la transacción. Si hay algún problema, se invoca *rollback*.

Transacciones distribuidas en la modalidad no gestionada de .NET

Las clases IBM MQ.NET admiten conexiones no gestionadas (cliente) que utilizan el cliente de transacciones ampliado y COM+/MTS como coordinador de transacciones, utilizando el modelo de programación de transacciones implícito o explícito. En la modalidad no gestionada, las clases IBM MQ .NET delegan todas sus llamadas al cliente de transacciones ampliado C que gestiona todo el proceso de transacciones en nombre de .NET.

El procesamiento de transacciones está controlado por un gestor de transacciones externo, que coordina la unidad de trabajo global bajo el control del API del gestor de transacciones. Los verbos MQBEGIN, MQCMIT y MQBACK no están disponibles.IBM MQ Las clases .NET exponen este soporte a través de su modalidad de transporte no gestionada (cliente C). Consulte Configuración de gestores de transacciones compatibles con XA

MTS ha evolucionado como sistema de proceso de transacciones (TP) para proporcionar las mismas características en Windows NT que están disponibles en CICS, Tuxedo y en otras plataformas. Cuando se instala el MTS, se añade un servicio independiente a Windows NT denominado Microsoft Distributed Transaction Coordinator (MSDTC). El MSDTC coordina las transacciones que abarcan distintos almacenes de datos o recursos. Para funcionar, requiere que cada almacén de datos implemente su propio gestor de recursos propietario.

IBM MQ se hace compatible con MSDTC implementando una interfaz (interfaz de gestor de recursos propietario) donde correlaciona las llamadas XA DTC con llamadas IBM MQ(X/Open). IBM MQ desempeña el rol de un gestor de recursos.

Cuando un componente como, por ejemplo, COM+ solicita acceso a IBM MQ, el COM normalmente comprueba con el objeto de contexto MTS apropiado si es necesaria una transacción. Si es necesaria una transacción, el COM informa al DTC e inicia automáticamente una transacción IBM MQ integral para esta operación. A continuación, COM trabaja con los datos a través del software MQMTS, colocando y recibiendo mensajes según sea necesario. La instancia de objeto obtenida de COM invoca los métodos SetComplete o SetAbort una vez terminadas todas las acciones sobre los datos. Cuando la aplicación invoca SetComplete, la llamada indica al DTC que la aplicación ha completado la transacción y que el DTC puede proseguir con el proceso de confirmación en dos fases. El DTC emite llamadas a MQMTS que, a su vez, llama a IBM MQ para confirmar o retrotraer la transacción.

Desarrollo de una aplicación IBM MQ .NET usando un cliente no gestionado

Para ejecutarse en el contexto de COM+, una clase .NET debe heredar del sistema .EnterpriseServices.ServicedComponent. Las reglas y recomendaciones para crear ensamblajes que utilicen componentes con servicio son las siguientes:

Nota: Los pasos siguientes solo son relevantes si se utiliza el modo System.EnterpriseServices.

- La clase y el método que se van a iniciar en COM+ tienen que ser públicos (nada de clases internas ni métodos protegidos o estáticos).
- Atributos de clase y método: El atributo TransactionOption dicta el nivel de transacción de la clase, es decir, si las transacciones están inhabilitadas, están soportadas o son necesarias. El atributo AutoComplete del método ExecuteUOW() indica a COM+ que confirme la transacción si no se lanza ninguna excepción no manejada.
- Nombrado fuerte de un ensamblaje: el ensamblaje ha de tener un nombrado fuerte y estar registrado en la caché de ensamblaje global (GAC). El ensamblaje se registra en COM+ explícitamente o de forma perezosa tras haberse registrado en la GAC.
- Registro de un ensamblaje en COM+: Prepare el ensamblaje para que se exponga a los clientes COM. A continuación, cree una biblioteca de tipos con la herramienta de registro de ensamblajes, regasm.exe.

```
regasm UnmanagedToManagedXa.dll
```

- Registre el ensamblaje en la GAC `gacutil /i UnmanagedToManagedXa.dll`.
- Registre el ensamblaje en COM+ utilizando la herramienta del programa de instalación de servicios de .NET, regsvcs.exe. Consulte la biblioteca de tipos creada por regasm.exe:

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- El ensamblaje se despliega en la GAC y después se registra en COM+ de forma perezosa. La infraestructura de .NET se ocupa del registro después de que se ejecute el código por primera vez.

El flujo del código de ejemplo que usa el modelo System.EnterpriseServices y System.Transactions con COM+ se describe en las secciones siguientes:

Código de ejemplo que usa el modelo System.EnterpriseServices

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.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();
}
```

Flujo del código de ejemplo que usa System.Transactions en interacciones con COM+

```
[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();
}
```

Creación de mensajes de colocación y obtención sencillos dentro de un TransactionScope (ámbito transaccional)

Hay aplicaciones de ejemplo en C# de producto disponibles en IBM MQ. Estas aplicaciones simples muestran cómo colocar y obtener mensajes dentro de un TransactionScope. Al final de la tarea, podrá colocar y obtener mensajes de una cola o un tema.

Antes de empezar

El servicio MSDTC se debe estar ejecutando y debe estar habilitado para las transacciones XA.

Acerca de esta tarea

El ejemplo es una aplicación sencilla, `SimpleXAPut` y `SimpleXAGet`. Los programas `SimpleXAPut` y `SimpleXAGet` son aplicaciones en C# disponibles en IBM MQ. `SimpleXAPut` ilustra la utilización de MQPUT en transacciones distribuidas utilizando el espacio de nombres SystemTransactions. `SimpleXAGet` ilustra la utilización de MQGET en transacciones distribuidas utilizando el espacio de nombres SystemTransactions.

`SimpleXAPut` se encuentra en `MQ\tools\dotnet\samples\cs\base`

Procedimiento

Las aplicaciones se pueden ejecutar con los parámetros de línea de comandos en `tools\dotnet\samples\cs\base\bin`

```
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]
```

donde los parámetros son:

-destinationURI

Puede ser una cola o un tema. Para una cola, especifique como `queue://queueName` y para un tema especifique como `topic://topicName`.

-host

Puede ser un nombre de host como, por ejemplo, localhost o una dirección IP.

-port

El puerto con el que ejecuta el gestor de colas.

-channel

Canal de conexión que se utiliza. El valor predeterminado es `SYSTEM.DEF.SVRCONN`

-transaction

El resultado de la transacción, por ejemplo, commit (confirmación) o rollback (retrotracción).

-mode

El modo de transporte, por ejemplo, managed (gestionado) o unmanaged (no gestionado).

-numberOfMsgs

El número de mensajes. El valor predeterminado es 1.

Ejemplo

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Recuperación de transacciones en IBM MQ .NET

En esta sección se describe el proceso de recuperación de transacciones en IBM MQ .NET XA utilizando la modalidad gestionada.

Acerca de esta tarea

En el proceso de transacciones distribuidas, las transacciones se pueden completar correctamente, pero puede haber casos en los que una transacción puede fallar por muchas razones. Estas razones pueden incluir una anomalía del sistema, una anomalía de hardware, un error de red, datos incorrectos o no válidos, errores de aplicación o desastres naturales o provocados por el hombre. Es imposible evitar completamente los errores de transacción. El sistema de transacciones distribuidas tiene que ser capaz de manejar estos fallos. Tiene que ser capaz de detectar y corregir errores cuando se produzcan. Este proceso se conoce como Recuperación de transacciones.

Un aspecto importante del procesamiento de transacciones distribuidas consiste en recuperar las transacciones incompletas o dudosas. Es fundamental ejecutar la recuperación mientras la unidad de trabajo de una determinada transacción se mantiene bloqueada hasta que se recupera. Microsoft.NET permite recuperar transacciones incompletas o dudosas en su biblioteca de clases System.Transactions.

Este soporte de recuperación espera del gestor de recursos que mantenga registros de las transacciones y ejecute la recuperación cuando sea necesario.

En el modelo de recuperación de transacción de Microsoft .NET, el gestor de transacciones (System.Transactions, o Microsoft Distributed Transaction Coordinator (MS DTC), o ambos), inicia, coordina y controla la recuperación de transacciones. Los gestores de recursos basados en el protocolo OLE Tx (el protocolo Microsoft XA) proporcionan las opciones para configurar el DTC para que impulse, coordine y controle la recuperación para ellos. Para ello, los gestores de recursos tienen que registrar XA_Switch en MS DTC mediante la interfaz nativa.

XA_Switch proporciona los puntos de entrada de las funciones XA como, por ejemplo, xa_start, xa_end y xa_recover en el gestor de recursos al coordinador de transacciones distribuidas (DTC).

Recuperación utilizando el Microsoft Distributed Transaction Coordinator (DTC):

Microsoft Distributed Transaction Coordinator proporciona dos tipos de proceso de recuperación:

Recuperación en frío

La recuperación en frío se lleva a cabo si el proceso del gestor transaccional falla mientras hay abierta una conexión con un gestor de recursos XA. Cuando se reinicia el gestor de transacciones, lee sus registros cronológicos y restablece la conexión con el gestor de recursos XA, y luego inicia la recuperación.

Recuperación en caliente

La recuperación en caliente se realiza si el gestor de transacciones permanece activo mientras falla su conexión con el gestor de recursos XA porque fallan este último o la red. Tras el fallo, el gestor de transacciones intenta periódicamente reconectarse con el gestor de recursos XA. Cuando se restablece la conexión, el gestor de transacciones inicia la recuperación XA.

El espacio de nombres de System.Transactions proporciona la implementación gestionada de transacciones distribuidas basadas en MS DTC como gestor de transacciones. Proporciona funciones similares a las de la interfaz nativa de MS DTC, pero en un entorno completamente gestionado. La única diferencia está en la recuperación de transacciones. System.Transactions espera de los gestores de recursos que se hagan cargo de la recuperación por sí mismos y luego coordina con los gestores de transacciones (MS DTC). El gestor de recursos tiene que solicitar la recuperación de una determinada transacción incompleta y luego el gestor de transacciones lo acepta y coordina basándose en el resultado real de dicha transacción.

Proceso de recuperación de transacciones para IBM MQ .NET

En esta sección se describe cómo se pueden recuperar las transacciones distribuidas con las clases IBM MQ .NET .

Visión general

Para recuperar una transacción incompleta, es necesaria la información de recuperación. Los gestores de colas deben registrar la información de recuperación de las transacciones en el almacenamiento.IBM MQ Las clases .NET siguen una vía de acceso similar. La información de recuperación de las transacciones se registra en una cola del sistema con el nombre SYSTEM.DOTNET.XARECOVERY.QUEUE.

La recuperación de transacciones en IBM MQ .NET es un proceso de dos etapas:

1. Registro de información de recuperación de transacciones en SYSTEM.DOTNET.XARECOVERY.QUEUE.
2. Recuperación de transacciones utilizando la aplicación XA Monitor WmqDotnetXAMonitor.

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.QUEUE es una cola del sistema que contiene información de recuperación de transacciones para transacciones incompletas. Esta cola se crea cuando se crea un gestor de colas.

Para cada transacción, durante la fase de preparación se añade a SYSTEM.DOTNET.XARECOVERY.QUEUE un mensaje persistente que contiene la información de recuperación. El mensaje se suprime si la llamada de confirmación se ejecuta correctamente.

Nota: No debe suprimir SYSTEM.DOTNET.XARECOVERY.QUEUE .

Aplicación WMQDotnetXAMonitor

IBM MQ .NET La aplicación Supervisor XA, WmqDotnetXAMonitor, es una aplicación gestionada de .NET que supervisa un gestor de colas y procesa mensajes en SYSTEM.DOTNET.XARECOVERY.QUEUE y recupera transacciones incompletas

Si el agente de canal de mensajes (MCA) no puede transferir el mensaje a la cola de destino, genera un informe de excepción que contiene el mensaje original y lo coloca en una cola de transmisión para enviarlo a la cola de respuesta especificada en el mensaje original. (Si la cola de respuesta se encuentra en el mismo gestor de colas que el MCA, el mensaje se transfiere directamente a esa cola, no a una cola de transmisión).

Las siguientes operaciones se consideran incompletas y se recuperan:

- Si la transacción se ha preparado pero no se ha completado COMMIT dentro del periodo de tiempo de espera.
- Si la transacción se ha preparado pero el gestor de colas de IBM MQ se ha cerrado.
- Si la transacción está preparado pero, a continuación, se ha cerrado el gestor de transacciones.

  La aplicación Supervisor XA debe ejecutarse desde el mismo sistema en el que se ejecuta la aplicación cliente de IBM MQ .NET . Si hay aplicaciones que se ejecutan en varios sistemas y se conectan al mismo gestor de colas, la aplicación XAMonitor WmqDotnetse debe ejecutar desde todos los sistemas. Aunque cada máquina cliente tiene una instancia de la aplicación XA Monitor en ejecución para recuperar la aplicación, cada instancia de XA Monitor debería poder identificar el mensaje que corresponde a la transacción que estaba coordinando el MS DTC local del supervisor XA actual para que pueda volver a incluirlo y completarlo.

Conceptos relacionados

"Casos de uso de recuperación de transacciones para IBM MQ .NET" en la página 591
Hay varios casos de uso diferentes en los que puede ser necesario recuperar una transacción.

Tareas relacionadas

"Utilización de la aplicación WMQDotnetXAMonitor" en la página 592
El cliente IBM MQ .NET proporciona una aplicación de Monitor XA, WmqDotnetXAMonitor, que se puede utilizar para recuperar cualquier transacción distribuida incompleta. La aplicación WmqDotnetXAMonitor establece una conexión con el gestor de colas donde las transacciones están pendientes y, a continuación, resuelve la transacción basándose en los parámetros que ha establecido.

Casos de uso de recuperación de transacciones para IBM MQ .NET

Hay varios casos de uso diferentes en los que puede ser necesario recuperar una transacción.

- **IBM MQ Aplicación que utiliza un único DTC y una única instancia de gestor de colas:** en este caso de uso, cuando se conecta al gestor de colas y ejecuta Unidad de trabajo (UoW) bajo la transacción, y si la transacción falla y pasa a estar incompleta, la aplicación Supervisor XA recupera la transacción y la completa.

  En este caso de uso, habrá una única instancia de la aplicación Supervisor XA en ejecución, ya que un único gestor de colas está asociado a las transacciones.
- **Varias aplicaciones IBM MQ que utilizan un único DTC y una única instancia de gestor de colas:** en este caso de uso, hay más de una aplicación IBM MQ bajo un único DTC y todas se conectan al mismo gestor de colas y ejecutan UoW bajo transacciones.

  Si las transacciones fallan y pasan a estar incompletas, la aplicación Supervisor XA las recupera y completa las transacciones que pertenecen a todas las aplicaciones.

En este caso de uso, se ejecuta una única instancia de la aplicación Supervisor XA, ya que se utiliza un gestor de colas en las transacciones.

- **Varias aplicaciones IBM MQ , varios DTC, distintas instancias de gestor de colas:** en este caso de uso, hay más de una aplicación IBM MQ bajo distintos DTC (es decir, cada aplicación se ejecuta en una máquina diferente) y se conecta a distintos gestores de colas.

  Si se produce un fallo y la transacción se vuelve incompleta, la aplicación supervisora comprueba el TransactionManagerWhereabouts del mensaje para determinar la dirección del DTC. Si el valor de TransactionManagerWhereabouts coincide con la dirección del DTC bajo el que ejecuta el supervisor, completa la recuperación; en caso contrario, sigue buscando hasta encontrar el mensaje que corresponde a su DTC.

  En este caso de uso, sólo habrá una instancia de la aplicación XA Monitor en ejecución por cliente (usuario o sistema), ya que cada cliente tiene su propio gestor de colas utilizado en las transacciones.

- **Varias aplicaciones IBM MQ , varios DTC, varias instancias del mismo gestor de colas:** en este caso de uso, hay más de una aplicación IBM MQ bajo diferentes DTC (es decir, cada aplicación se ejecuta en una máquina diferente) y todas se conectan al mismo gestor de colas.

  Si se produce un fallo y la transacción se vuelve incompleta, la aplicación supervisora verifica el TransactionManagerWhereabouts del mensaje para comprobar si la dirección y el valor de DTC coinciden con el DTC bajo el que se ejecuta el supervisor. Si ambos valores coinciden, completa la recuperación; en caso contrario, sigue buscando hasta encontrar el mensaje correspondiente a su DTC.

  En este caso de uso, sólo habrá una única instancia de la aplicación Supervisor XA en ejecución por cliente (usuario o sistema), ya que cada cliente tiene su propia asociación de gestor de colas utilizada en las transacciones.

- **Varias aplicaciones IBM MQ , DTC único, distintas instancias de gestor de colas:** en este caso de uso, hay más de una aplicación IBM MQ bajo un único DTC (es decir, en un sistema, hay más de una aplicación IBM MQ en ejecución) y que se conecta a distintos gestores de colas.

  Si la transacción falla y se vuelve incompleta, la aplicación de supervisión recupera la transacción.

  En este caso de uso, habrá tantas instancias de aplicación de supervisión ejecutando como gestores de colas con los que se conecte, ya que cada aplicación usa su propio gestor de colas en las transacciones y hay que recuperar cada uno de ellos.

  Nota: Si la aplicación de supervisión XA no se está ejecutando en segundo plano, puede iniciarla.

Conceptos relacionados

"Proceso de recuperación de transacciones para IBM MQ .NET" en la página 590
En esta sección se describe cómo se pueden recuperar las transacciones distribuidas con las clases IBM MQ .NET .

Tareas relacionadas

"Utilización de la aplicación WMQDotnetXAMonitor" en la página 592
El cliente IBM MQ .NET proporciona una aplicación de Monitor XA, WmqDotnetXAMonitor, que se puede utilizar para recuperar cualquier transacción distribuida incompleta. La aplicación WmqDotnetXAMonitor establece una conexión con el gestor de colas donde las transacciones están pendientes y, a continuación, resuelve la transacción basándose en los parámetros que ha establecido.

Utilización de la aplicación WMQDotnetXAMonitor

El cliente IBM MQ .NET proporciona una aplicación de Monitor XA, WmqDotnetXAMonitor, que se puede utilizar para recuperar cualquier transacción distribuida incompleta. La aplicación WmqDotnetXAMonitor establece una conexión con el gestor de colas donde las transacciones están pendientes y, a continuación, resuelve la transacción basándose en los parámetros que ha establecido.

Acerca de esta tarea

La aplicación WMQDotnetXAMonitor se debe ejecutar manualmente. Puede iniciarse en cualquier momento. Puede iniciarlo cuando vea los mensajes en SYSTEM.DOTNET.XARECOVERY.QUEUE o puede

mantenerlo en ejecución en segundo plano antes de realizar cualquier trabajo transaccional con las aplicaciones que se escriben utilizando clases IBM MQ .NET .

Puede establecer los valores de parámetro para WMQDotnetXAMonitor a través de la línea de mandatos o utilizando un archivo de configuración de aplicación. Los valores que se proporcionan a través del archivo de configuración de la aplicación tienen prioridad sobre los valores establecidos a través de la línea de mandatos.

Antes de IBM MQ 9.3.0, la conexión que WMQDotnetXAMonitor establece es una conexión no segura.

V 9.3.0 A partir de IBM MQ 9.3.0, tiene la opción de establecer una conexión segura con el gestor de colas estableciendo parámetros adicionales para WMQDotnetXAMonitor.

Procedimiento

- Para proporcionar entrada a WmqDotNETXAMonitor utilizando un archivo de configuración de aplicación, consulte "Valores del archivo de configuración de la aplicación WmqDotNETXAMonitor" en la página 594.
- Para iniciar la aplicación WMQDotnetXAMonitor desde la línea de mandatos, utilice el mandato siguiente con los parámetros que necesite:

  Antes IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  V 9.3.0 Desde IBM MQ 9.3.0:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  Los parámetros que puede especificar son los siguientes:

  – **-m *NombreGestorColas***
  Nombre del gestor de colas.

  Opcional

  -n *NombreConexión*
  El nombre de conexión en formato de host (puerto). *ConnectionName* puede contener más de un nombre de conexión. Se deben proporcionar varios nombres de conexión en una lista separada por comas, por ejemplo `localhost (1414), localhost (1415), localhost (1416)`. La aplicación WMQDotnetXAMonitor ejecuta la recuperación para cada uno de los nombres de conexión especificados en la lista separada por comas.

  -c *ChannelName*
  El nombre de canal.

  -i
  Finalización de bifurcación heurística.

  Opcional

  V 9.3.0 **-k *Repositorio de claves SSL***
  El nombre del repositorio de claves SSL. Los valores soportados son:

  - *SYSTEM (es el valor por omisión)
  - *XX_ENCODE_CASE_ONE usuario

  Opcional

  V 9.3.0 **-s *Especificación de cifrado***
  La CipherSpec que establezca debe ser una de las CipherSpecs para la versión soportada y preferiblemente puede ser la misma que la especificada en la política de grupo de

Windows . Para obtener más información, consulte “Soporte de CipherSpec para el cliente .NET gestionado” en la página 616.

Obligatorio para establecer una conexión segura con el gestor de colas.

V 9.3.0 **-dn *Nombre de SSLPeer***

El nombre de igual SSL utilizado para comprobar el nombre distinguido (DN) del certificado del gestor de colas de igual.

Opcional

V 9.3.0 **-cl *Etiqueta de certificado***

El nombre de etiqueta que identifica el certificado.

Opcional

V 9.3.0 **-sn *OutboundSNI***

Indica si la indicación de nombre de servidor (SNI) debe establecerse en el nombre de canal IBM MQ de destino en el sistema remoto al iniciar una conexión TLS, o en el nombre de host. Los valores soportados para esta opción son:

- CHANNEL (es el valor predeterminado)
- HOSTNAME
- *

Si no se establece ningún valor, se utiliza el valor predeterminado, es decir, `CHANNEL`.

Opcional

V 9.3.0 **-cr *Comprobación de revocación de certificados***

Si se debe realizar la comprobación de revocación de certificación. Los valores soportados para esta opción son:

- true
- false (es el valor predeterminado)

Opcional

V 9.3.0 **-kr *KeyResetKeyReset***

El número total de bytes no cifrados que se envían y reciben en el canal antes de que se renegocie la clave secreta utilizada para el cifrado.

El valor predeterminado de 0 indica que las claves secretas nunca se renegocian

Opcional

La aplicación WMQDotnetXAMonitor realiza las acciones siguientes:

1. Comprueba la profundidad de cola de SYSTEM.DOTNET.XARECOVERY.QUEUE con un intervalo de 100 segundos.
2. Si la profundidad de cola es mayor que cero, examina la cola en busca de mensajes y comprueba si los mensajes satisfacen los criterios de transacción incompletos.
3. Si un mensaje cumple los criterios de transacción incompletos, lo extrae y recupera la información de recuperación de la transacción.
4. Determina si la información de recuperación está relacionada con el Microsoft Coordinador de transacciones distribuidas local (MS DTC). Si este es el caso, WMQDotnetXAMonitor continúa recuperando la transacción; de lo contrario, vuelve a examinar el siguiente mensaje.
5. Realiza llamadas al gestor de colas para recuperar la transacción incompleta.

Valores del archivo de configuración de la aplicación WmqDotNETXAMonitor

Puede proporcionar entrada a la aplicación IBM MQ .NET XA Monitor, WmqDotNETXAMonitor, utilizando un archivo de configuración de aplicación. Con IBM MQ .NET, se suministra un archivo de configuración de aplicación de ejemplo. Puede modificar este archivo de ejemplo de acuerdo con sus requisitos.

Los valores de entrada proporcionados a través del archivo de configuración de la aplicación tienen la prioridad más alta. Si proporciona valores de entrada en la línea de mandatos tal como se describe en "Utilización de la aplicación WMQDotnetXAMonitor" en la página 592 y en el archivo de configuración de la aplicación, los valores del archivo de configuración de la aplicación tienen prioridad.

Archivo de configuración de aplicación de ejemplo para antes de IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

V 9.3.0 Archivo de configuración de aplicación de ejemplo de IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

Registro de aplicación WmqDotNetXAMonitor

La aplicación de supervisión crea un archivo de registro en el directorio de aplicación para registrar su progreso y el estado de recuperación de las transacciones. El registro se inicia con el nombre de conexión y los detalles del canal para mostrar el gestor de colas actual cuya recuperación está ejecutando.

Una vez que se inicia la recuperación, se registran el MessageId del mensaje de recuperación de transacción, el TransactionId de la transacción incompleta y el resultado de la transacción según la coordinación de Transaction Manager.

Archivo de registro de ejemplo:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
```

```
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

Escritura y despliegue de programas de IBM MQ .NET

Para utilizar IBM MQ classes for .NET para acceder a colas de IBM MQ, escriba programas en cualquier lenguaje soportado por .NET que contengan llamadas que coloquen mensajes en y obtengan mensajes de colas IBM MQ.

La documentación de IBM MQ contiene información sólo para los lenguajes C#, C++ y Visual Basic.

Esta colección de temas proporciona información para ayudarle a crear aplicaciones que interactúan con sistemas IBM MQ. Para obtener detalles de clases individuales, consulte Las clases e interfaces de IBM MQ .NET.

Diferencias de conexión

La forma de programar para IBM MQ.NET tiene algunas dependencias respecto a las modalidades de conexión que desee utilizar.

Cuando IBM MQ classes for .NET se utiliza como cliente gestionado, existen varias diferencias respecto a un IBM MQ MQI client estándar, pues algunas funciones no están disponibles para un cliente gestionado.

IBM MQ.NET determina qué tipo de conexión utilizar a partir de los valores que especifique para el nombre de conexión, nombre de canal, el valor de personalización NMQ_MQ_LIB y la propiedad MQC.TRANSPORT_PROPERTY.

Conexiones de cliente gestionado

Cuando se utiliza IBM MQ classes for .NET como un cliente gestionado, hay varias diferencias respecto a un IBM MQ MQI client estándar.

Las siguientes características no están disponibles para un cliente gestionado:

- Compresión de canales
- Encadenamiento de salidas de canal

Si intenta utilizar estas características con un cliente gestionado, devolverá una MQException. Si el error se detecta en el extremo de cliente de una conexión, utilizará el código de razón MQRC_ENVIRONMENT_ERROR. Si se detecta en el extremo del servidor, se utilizará el código de razón devuelto por el servidor.

Las salidas de canal escritas para un cliente no gestionado no funcionan. Debe escribir nuevas salidas específicamente para el cliente gestionado. Compruebe que no haya salidas de canal no válidas especificadas en la tabla de definición de canal de cliente (CCDT).

El nombre de una salida de canal gestionado puede tener hasta 999 caracteres de largo. No obstante, si utiliza CCDT para especificar el nombre de salida de canal, está limitado a 128 caracteres.

La comunicación solo está soportada a través de TCP/IP.

Cuando detiene un gestor de colas utilizando el mandato **endmqm**, un canal de conexión de servidor a un cliente gestionado de .NET puede tardar más tiempo en cerrarse que los canales de conexión de servidor a otros clientes.

Si ha establecido *NMQ_MQ_LIB* en `managed` para poder utilizar los diagnósticos de problemas de IBM MQ gestionados, no se da soporte a ninguno de los parámetros -i, -p, -s, -b o -c del mandato **strmqtrc**.

Una aplicación de .NET gestionada que utiliza transacciones XA no funcionará con un gestor de colas de z/OS. Un cliente de .NET gestionado que intenta conectarse a un gestor de colas de z/OS falla con un error MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) en la llamada MQOPEN. No obstante, una aplicación de .NET gestionada que utiliza transacciones XA funcionará con el gestor de colas distribuido.

Definición del tipo de conexión a utilizar

El tipo de conexión se determina estableciendo el nombre de conexión, el nombre de canal, el valor de personalización NMQ_MQ_LIB y la propiedad MQC.TRANSPORT_PROPERTY.

Puede especificar el nombre de conexión de la siguiente manera:

- De forma explícita en un constructor de MQQueueManager:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Estableciendo las propiedades MQC.HOST_NAME_PROPERTY y, opcionalmente, MQC.PORT_PROPERTY en una entrada de tabla hash en un constructor de MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Como valores de MQEnvironment explícitos

```
MQEnvironment.Hostname
```

  `MQEnvironment.Port` (opcional).
- Estableciendo las propiedades MQC.HOST_NAME_PROPERTY y, opcionalmente, MQC.PORT_PROPERTY en la tabla hash MQEnvironment.properties.

Puede especificar el nombre de canal de la siguiente manera:

- De forma explícita en un constructor de MQQueueManager:

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- Estableciendo la propiedad MQC.CHANNEL_PROPERTY en una entrada de tabla hash en un constructor de MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Como un valor de MQEnvironment explícito

```
MQEnvironment.Channel
```

- Estableciendo la propiedad MQC.CHANNEL_PROPERTY en la tabla hash MQEnvironment.properties.

Puede especificar la propiedad de transporte de la siguiente manera:

- Estableciendo la propiedad MQC.TRANSPORT_PROPERTY en una entrada de tabla hash en un constructor de MQQueueManager:

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Estableciendo la propiedad MQC.TRANSPORT_PROPERTY en la tabla hash MQEnvironment.properties.

Seleccione el tipo de conexión que necesite utilizando uno de los valores siguientes:

MQC.TRANSPORT_MQSERIES_BINDINGS - conectar como servidor
MQC.TRANSPORT_MQSERIES_CLIENT - conectar como cliente no XA

MQC.TRANSPORT_MQSERIES_XACLIENT - conectar como cliente XA
MQC.TRANSPORT_MQSERIES_MANAGED - conectar como cliente gestionado no XA

Puede establecer el valor de personalización NMQ_MQ_LIB para elegir de forma explícita el tipo de conexión según se muestra en la tabla siguiente.

Valor de NMQ_MQ_LIB	**Tipo de conexión**
mqic.dll	Conectar como un cliente no XA
mqicxa.dll	Conectar como un cliente XA
mqm.dll	Conectar como un servidor o como un cliente no XA
gestionado	Conectar como un cliente gestionado no XA

Nota: Los valores de mqic32.dll y mqic32xa.dll se aceptan como sinónimos de mqic.dll y mqicxa.dll por compatibilidad con releases anteriores. No obstante, mqm.dll y mqm.pdb solo forman parte del paquete de cliente de IBM WebSphere MQ 7.1 en adelante.

Si elige un tipo de conexión que no está disponible en el entorno, por ejemplo, especifica mqic32xa.dll y no tiene soporte XA, IBM MQ.NET lanzará una excepción.

El establecimiento de NMQ_MQ_LIB en "managed" provoca que el cliente utilice pruebas de diagnóstico de problemas de IBM MQ gestionadas, conversión de datos .NET, y otras funciones de IBM MQ de bajo nivel gestionadas.

Todos los demás valores de NMQ_MQ_LIB provocan que el proceso .NET utilice conversión de datos y pruebas de diagnóstico de problemas de IBM MQ sin gestionar, y otras funciones de IBM MQ de bajo nivel no gestionadas (suponiendo que hay un servidor o un IBM MQ MQI client instalado en el sistema).

IBM MQ.NET elige el tipo de conexión de la siguiente manera:

1. Si se especifica MQC.TRANSPORT_PROPERTY, se conecta en función del valor de MQC.TRANSPORT_PROPERTY.

   Tenga en cuenta, no obstante, que el establecimiento de MQC.TRANSPORT_PROPERTY en MQC.TRANSPORT_MQSERIES_MANAGED no garantiza que el proceso del cliente se ejecute como gestionado. Incluso con este valor, el cliente no estará gestionado en los casos siguientes:

   - Si otra hebra del proceso se ha conectado con MQC.TRANSPORT_PROPERTY establecido en algo diferente a MQC.TRANSPORT_MQSERIES_MANAGED.
   - Si NMQ_MQ_LIB no se establece en "managed", las pruebas de diagnóstico de problemas, la conversión de datos y otras funciones de bajo nivel no estarán completamente gestionadas (suponiendo que hay un servidor o un IBM MQ MQI client instalado en el sistema).
2. Si se ha especificado un nombre de conexión sin un nombre de canal, o se ha especificado un nombre de canal sin un nombre de conexión, se lanzará un error.
3. Si se han especificado tanto un nombre de conexión como un nombre de canal:
   - Si NMQ_MQ_LIB se establece en mqic32xa.dll, se conecta como un cliente XA.
   - Si NMQ_MQ_LIB se establece en managed, se conecta como un cliente gestionado.
   - En cualquier otro caso, se conecta como un cliente no XA.
4. Si se especifica NMQ_MQ_LIB, se conecta en función del valor de NMQ_MQ_LIB.
5. Si hay un servidor IBM MQ instalado, se conecta como un servidor.
6. Si hay un IBM MQ MQI client instalado, se conecta como un cliente no XA.
7. En cualquier otro caso, se conecta como un cliente gestionado.

Windows Utilización de la plantilla de proyecto de IBM MQ .NET

El cliente de IBM MQ .NET le ofrece la posibilidad de utilizar una plantilla de proyecto para ayudarle a desarrollar sus aplicaciones .NET Core .

Antes de empezar

Debe tener Microsoft Visual Studio 2017, o posterior, y .NET Core 2.1 en el sistema.

Debe copiar la plantilla .NET de la

`&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip`

al directorio

`&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates`

directorio, donde:

- *& MQ_INSTALL_ROOT* es el directorio raíz de la instalación
- *& USER_HOME_DIRECTORY* es el directorio de inicio.

Debe detener y reiniciar Microsoft Visual Studio para seleccionar la plantilla.

Acerca de esta tarea

La plantilla de proyecto .NET incluye algún código común que puede utilizar para ayudar a desarrollar las aplicaciones. Con el código incorporado, puede conectarse al gestor de colas IBM MQ y realizar una operación de colocación u obtención simplemente modificando las propiedades del código incorporado.

Procedimiento

1. Abra Microsoft Visual Studio.
2. Pulse en **Archivo**, seguido por **Nuevo** y, a continuación, **Proyecto**.
3. En la ventana *Crear un proyecto nuevo*, seleccione `IBM MQ .NET Client App (.NET Core)` y pulse **Siguiente**.
4. En la ventana *Configurar el proyecto nuevo*, cambie el *Nombre de proyecto* del proyecto si lo desea y pulse **Crear** para crear el proyecto .NET.

   `MQDotnetApp.cs` es el archivo que se crea junto con el archivo de proyecto. Este archivo contiene el código que se conecta al gestor de colas y realiza una operación de colocar y obtener.

   Las propiedades de conexión se establecen en valores predeterminados:

   - MQC.CONNECTION_NAME_PROPERTY se establece en *localhost(1414)*
   - MQC.CHANNEL_PROPERTY se establece en *DOTNET.SVRCONN*

   La cola se establece en *Q1*, y puede modificar estas propiedades en consecuencia.
5. Compile y ejecute la aplicación.

Conceptos relacionados

Componentes y características de IBM MQ

Tiempo de ejecución de la aplicación .NET - Solamente en Windows

Archivos de configuración para IBM MQ classes for .NET

Una aplicación cliente de .NET puede utilizar un archivo de configuración de IBM MQ MQI client y, si está utilizando el tipo de conexión gestionada, un archivo de configuración de aplicación de .NET. Los valores contenidos en el archivo de configuración de aplicación tienen prioridad.

Archivo de configuración de cliente

Una aplicación cliente de IBM MQ classes for .NET puede utilizar un archivo de configuración de cliente de la misma forma que cualquier otro IBM MQ MQI client. Este archivo normalmente se denomina `mqclient.ini`, pero puede especificar un nombre de archivo diferente. Para obtener más información sobre el archivo de configuración de cliente, consulte el archivo de configuración de IBM MQ MQI client , `mqclient.ini`.

Sólo los atributos siguientes contenidos en un archivo de configuración de IBM MQ MQI client son válidos para IBM MQ classes for .NET. Si especifica otros atributos, esta acción no tendrá afecto.

Tabla 77. Atributos de archivo de configuración de cliente que son relevantes para IBM MQ classes for .NET

Stanza	Atributo
canales	CCSID
canales	ChannelDefinitionDirectory
canales	ChannelDefinitionFile
canales	ReconDelay
canales	DefRecon
canales	MQReconnectTimeout
canales	ServerConnectionParms
canales	Put1DefaultAlwaysSync
canales	PasswordProtection
ClientExitPath	Vía de acceso predeterminada de las salidas
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
V 9.3.0 V 9.3.0 Seguridad	Archivo MQIInitialKey
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository
V 9.3.0 V 9.3.0 SSL	Contraseña de SSLKeyRepository
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

Puede alterar temporalmente cualquiera de estos atributos utilizando la variable de entorno adecuada.

Archivo de configuración de aplicación

Si está utilizando el tipo de conexión gestionada, también puede alterar temporalmente el archivo de configuración de cliente de IBM MQ y las variables de entorno equivalentes utilizando el archivo de configuración de aplicación de .NET.

Los valores del archivo de configuración de aplicación de .NET solo se aplican cuando se utiliza el tipo de conexión gestionada, y no se tienen en cuenta para otros tipos de conexión.

El archivo de configuración de aplicación de .NET y su formato están definidos por Microsoft para su uso general dentro de la infraestructura de .NET, pero los nombres de sección, las claves y los valores específicos mencionados en esta documentación son específicos de IBM MQ.

El formato del archivo de configuración de aplicación de .NET consta de varias *secciones*. Cada sección contiene una o más *claves*, y cada clave tiene un *valor* asociado. El ejemplo siguiente muestra las secciones, claves y valores utilizados en un archivo de configuración de aplicación de .NET para controlar la propiedad KeepAlive de TCP/IP:

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Las palabras clave utilizadas en los nombres de sección y claves del archivo de configuración de aplicación de .NET coinciden exactamente con las palabras clave correspondientes a las stanzas y atributos definidos en el archivo de configuración de cliente.

La sección <configSections> debe ser el primer elemento hijo del elemento <configuration>.

Consulte la documentación de Microsoft para obtener más información.

Fragmento de código C# de ejemplo para ser utilizado con .NET

Fragmento de código C# que muestra una aplicación que se conecta a un gestor de colas, pone un mensaje en la cola y recibe una respuesta.

El fragmento de código C# de ejemplo muestra una aplicación que realiza tres acciones:

1. Conectarse a un gestor de colas
2. Transferir un mensaje a SYSTEM.DEFAULT.LOCAL.QUEUE
3. Obtener de nuevo el mensaje

También muestra como cambiar el tipo de conexión.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";
```

```
/// <summary>
/// Initialise the connection properties for the connection type requested
/// </summary>
/// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
static Hashtable init(String connectionType)
{
  Hashtable connectionProperties = new Hashtable();

  // Add the connection type
  connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

  // Set up the rest of the connection properties, based on the
  // connection type requested
  switch(connectionType)
  {
    case MQC.TRANSPORT_MQSERIES_BINDINGS:
      break;
    case MQC.TRANSPORT_MQSERIES_CLIENT:
    case MQC.TRANSPORT_MQSERIES_XACLIENT:
    case MQC.TRANSPORT_MQSERIES_MANAGED:
      connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
      connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
      break;
  }

  return connectionProperties;
}
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static int Main(string[] args)
{
  try
  {
    Hashtable connectionProperties = init(connectionType);

    // Create a connection to the queue manager using the connection
    // properties just defined
    MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

    // Set up the options on the queue we want to open
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

    // Now specify the queue that we want to open,and the open options
    MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

    // Define an IBM MQ message, writing some text in UTF format
    MQMessage hello_world = new MQMessage();
    hello_world.WriteUTF("Hello World!");

    // Specify the message options
    MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                         // same as MQPMO_DEFAULT

    // Put the message on the queue
    system_default_local_queue.Put(hello_world, pmo);



    // Get the message back again

    // First define an IBM MQ message buffer to receive the message
    MQMessage retrievedMessage =new MQMessage();
    retrievedMessage.MessageId =hello_world.MessageId;

    // Set the get message options
    MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                        //same as MQGMO_DEFAULT

    // Get the message off the queue
    system_default_local_queue.Get(retrievedMessage,gmo);

    // Prove we have the message by displaying the UTF message text
    String msgText = retrievedMessage.ReadUTF();
    Console.WriteLine("The message is: {0}", msgText);

    // Close the queue
    system_default_local_queue.Close();
```

```
      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

Configuración del entorno de IBM MQ

Antes de utilizar la conexión de cliente para conectar con gestor de colas, debe configurar el entorno de IBM MQ.

Nota: Este paso no es necesario cuando se utiliza IBM MQ classes for .NET en la modalidad de enlaces de servidor.

La interfaz de programación de .NET le permite utilizar el valor de personalización NMQ_MQ_LIB, pero también incluye una clase MQEnvironment. Esta clase le permite especificar datos que se deben utilizar durante el intento de conexión, tales como los siguientes:

- Nombre de canal
- Nombre de host
- Número de puerto
- Salidas de canal
- Parámetros de SSL
- ID de usuario y contraseña

Para obtener información completa sobre la clase MQEnvironment, consulte Clase MQEnvironment.NET

Para especificar el nombre de canal y el nombre de host, utilice el código siguiente:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

De forma predeterminada, los clientes intentan conectar con un escucha de IBM MQ situado en el puerto 1414. Para especificar otro puerto, utilice el código:

```
MQEnvironment.Port = nnnn;
```

Conexión y desconexión de un gestor de colas

Cuando haya configurado el entorno de IBM MQ, estará listo para conectar con un gestor de colas.

Para conectar con un gestor de colas, cree una nueva instancia de la clase MQQueueManager:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Para desconectar de un gestor de colas, llame al método `Disconnect` en el gestor de colas:

```
queueManager.Disconnect();
```

Debe tener autorización para inquire (`inq`) en el gestor de colas para conectar con el gestor de colas. Sin autorización para inquire, el intento de conexión falla.

Si llama al método `Disconnect`, se cierran todas las colas y procesos abiertos a los que ha accedido a través del gestor de colas. Pero es una práctica de programación recomendada cerrar esos recursos explícitamente cuando termine de utilizarlos. Para cerrar los recursos, utilice el método `Close` en el objeto asociado a cada recurso.

Los métodos `Commit` y `Backout` en un gestor de colas sustituyen a las llamadas MQCMIT y MQBACK que se utilizan con la interfaz orientada a procedimientos.

Acceso a colas y temas

Puede acceder a las colas y los temas utilizando los métodos de MQQueueManager o los constructores correspondientes.

Para acceder a las colas, utilice los métodos de la clase MQQueueManager. MQOD (estructura de descriptor de objeto) se ha contraído en los parámetros de estos métodos. Por ejemplo, para abrir una cola en un gestor de colas representado por un objeto MQQueueManager denominado queueManager, utilice el código siguiente:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

El parámetro *options* es igual que el parámetro Options de la llamada MQOPEN.

El método AccessQueue devuelve un nuevo objeto de la clase MQQueue.

Cuando haya terminado de utilizar la cola, utilice el método Close() para cerrarla, tal como se muestra en el ejemplo siguiente:

```
queue.Close();
```

Con IBM MQ .NET, también puede crear una cola utilizando el constructor MQQueue. Los parámetros son exactamente los mismos que para el método accessQueue, con la adición de un parámetro de gestor de colas que especifica el objeto MQQueueManager para el que se ha creado una instancia que se utiliza. Por ejemplo:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

La creación de un objeto de cola utilizando este método permite escribir sus propias subclases de MQQueue.

De forma parecida, también puede acceder a los temas utilizando los métodos de la clase MQQueueManager. Utilice un método AccessTopic() para abrir un tema. Se devuelve un nuevo objeto de la clase MQTopic. Cuando haya terminado de utilizar el tema, utilice el método Close() de MQTopic para cerrarlo.

También puede crear un tema utilizando un constructor MQTopic. Hay una serie de constructores para los temas: para obtener más información, consulte Clase MQTopic.NET.

Manejo de mensajes

Los mensajes se manejan utilizando métodos de las clases de cola o de tema. Para crear un mensaje nuevo, cree un nuevo objeto MQMessageobject.

Los mensajes se transfieren a colas o temas mediante el método Put() de la clase MQQueue o MQTopic. Los mensajes se obtienen de las colas o temas mediante el método Get() de la clase MQQueue o MQTopic. A diferencia de la interfaz orientada a procedimientos, donde MQPUT y MQGET transfieren y obtienen matrices de bytes, IBM MQ classes for .NET transfiere y obtiene instancias de la clase MQMessage. La clase MQMessage encapsula el almacenamiento intermedio de datos que contiene los datos reales de los mensajes, junto con todos los parámetros MQMD (descriptor de mensaje) que describen dicho mensaje.

Para crear un mensaje nuevo, cree una nueva instancia de la clase MQMessage y utilice los métodos WriteXXX para colocar datos en el almacenamiento intermedio de mensajes.

Cuando se crea la instancia de mensaje nueva, todos los parámetros MQMD se establecen automáticamente en sus valores predeterminados, tal como se define en Valores iniciales y declaraciones de lenguaje para MQMD. El método Put() de MQQueue también toma una instancia de la clase MQPutMessageOptions como parámetro. Esta clase representa la estructura MQPMO. En el ejemplo siguiente se crea un mensaje y se transfiere a una cola:

```
// 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);
```

El método Get() de MQQueue devuelve una instancia nueva de MQMessage, que representa el mensaje recién obtenido de la cola. También toma una instancia de la clase MQGetMessageOptions como parámetro. Esta clase representa la estructura MQGMO.

No es necesario que especifique un tamaño máximo de mensaje, pues el método Get() ajusta automáticamente el tamaño de su almacenamiento intermedio interno para dar cabida al mensaje entrante. Utilice los métodos ReadXXX de la clase MQMessage para acceder a los datos del mensaje devuelto.

El ejemplo siguiente muestra cómo obtener un mensaje de una cola:

```
// 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();
```

Puede modificar el formato numérico que utilizan los métodos de lectura y escritura estableciendo la variable de miembro *encoding*.

Puede modificar el juego de caracteres que se va a utilizar para leer y escribir series de caracteres estableciendo la variable de miembro *characterSet*.

Consulte Clase MQMessage.NET para obtener más detalles.

Nota: El método WriteUTF() de MQMessage codifica automáticamente la longitud de la serie así como los bytes Unicode que contiene. Cuando el mensaje será leído por otro programa de .NET (mediante ReadUTF()), esta es la forma más sencilla de enviar información de tipo serie.

Manejo de las propiedades de mensaje

Las propiedades de mensaje le permiten seleccionar mensajes o recuperar información sobre un mensaje sin acceder a sus cabeceras. La clase MQMessage contiene métodos para obtener y establecer propiedades.

Puede utilizar las propiedades de mensaje para permitir que una aplicación seleccione los mensajes que se deben procesar o para recuperar información sobre un mensaje sin acceder a las cabeceras MQMD o MQRFH2. Las propiedades de mensaje también facilitan la comunicación entre las aplicaciones IBM MQ y JMS. Para obtener más información sobre las propiedades de mensaje en IBM MQ, consulte Propiedades de mensaje.

La clase MQMessage proporciona varios métodos para obtener y establecer propiedades, de acuerdo con el tipo de datos de la propiedad. Los métodos get utilizan nombres con el formato Get*Property y los métodos set utilizan nombres con el formato Set*Property, donde el asterisco (*) representa una de las series siguientes:

- Boolean
- Byte
- Bytes
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Largo
- Objeto
- Short
- Serie

Por ejemplo, para obtener la propiedad myproperty de IBM MQ (una serie de caracteres, utilice la llamada `message.GetStringProperty('myproperty')`. Opcionalmente puede pasar un descriptor de propiedad, que IBM MQ completará.

Manejo de errores

Manejo de errores procedentes de IBM MQ classes for .NET utilizando los bloques de código `try` y `catch`.

Los métodos de la interfaz de .NET no devuelven un código de terminación y un código de razón. En lugar de ello, emiten una excepción siempre que el código de terminación y el código de razón resultantes de una llamada a IBM MQ sean ambos distintos de cero. Esto simplifica la lógica del programa para que el usuario no necesite comprobar los códigos de retorno después de cada llamada a IBM MQ. Puede decidir en qué puntos del programa desea tratar la posibilidad de anomalía. En los puntos indicados, puede rodear el código con bloques `try` y `catch`, tal como se muestra en el ejemplo siguiente:

```
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 );
}
```

Obtención y establecimiento de valores de atributo

Las clases MQManagedObject, MQQueue y MQQueueManager contienen métodos que le permiten obtener y establecer sus valores de atributo. Observe que para MQQueue, los métodos sólo son efectivos si especifica los distintivos de inquire y set apropiados al abrir la cola.

Para los atributos comunes, las clases MQQueueManager y MQQueue heredan de una clase llamada MQManagedObject. Esta clase define las interfaces Inquire () y Set ().

Cuando crea un nuevo objeto de gestor de colas utilizando el operador *new*, el objeto se abre automáticamente para inquire. Cuando utiliza el método AccessQueue () para acceder a un objeto de cola, ese objeto no se abre automáticamente para las operaciones inquire ni set; esto podría causar problemas con algunos tipos de colas remotas. Para utilizar los métodos Inquire y Set y establecer propiedades en una cola, debe especificar los distintivos de inquire y set adecuados en el parámetro openOptions del método AccessQueue ().

Los métodos inquire y set tienen tres parámetros:

- matriz selectors
- matriz intAttrs
- matriz charAttrs

No necesita los parámetros SelectorCount, IntAttrCount y CharAttrLength que se encuentran en MQINQ, pues la longitud de una matriz se conoce siempre. El ejemplo siguiente muestra cómo efectuar una consulta sobre una cola:

```
//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);
```

Se pueden consultar todos los atributos de estos objetos. Un subconjunto de atributos se expone como las propiedades de un objeto. Para obtener una lista de atributos de objeto, consulte Atributos de objetos . Para conocer las propiedades de objeto, consulte la descripción de clase apropiada.

Programas multihebra

El entorno de ejecución .NET es multihebra de forma intrínseca. IBM MQ classes for .NET permite que varias hebras compartan un objeto de gestor de colas, pero asegura la sincronización de los accesos al gestor de colas de destino.

Considere por ejemplo un programa simple que se conecta a un gestor de colas y abre una cola durante el proceso de inicio. El programa visualiza un solo botón en la pantalla y, cuando el usuario lo pulsa, el programa busca un mensaje en la cola y lo carga. En esta situación, la inicialización de la aplicación se produce en una hebra, y el código que se ejecuta en respuesta a la pulsación del botón se ejecuta en una hebra separada (la hebra de la interfaz de usuario).

La implementación de IBM MQ .NET asegura, para una conexión determinada, (instancia de objeto MQQueueManager), que se sincronice todo el acceso al gestor de colas de IBM MQ. El comportamiento predeterminado es que una hebra que desee emitir una llamada a un gestor de colas se bloquea hasta que se completen todas las demás llamadas en curso para esa conexión. Si necesita acceso simultáneo al mismo gestor de colas desde varias hebras dentro del programa, cree un nuevo objeto MQQueueManager para cada hebra que necesite acceso simultáneo. (Equivale a emitir una llamada MQCONN independiente para cada hebra).

Si MQC.MQCNO_HANDLE_SHARE_NONE o MQC.MQCNO_SHARE_NO_BLOCK alteran temporalmente las opciones de conexión predeterminadas, el gestor de colas ya no está sincronizado.

Utilización de una tabla de definiciones de canal de cliente con .NET

Puede utilizar una tabla de definiciones de canal de cliente (CCDT) con IBM MQ classes for .NET. Puede especificar la ubicación de la tabla CCDT de diversas maneras, dependiendo de si está utilizando una conexión gestionada o no gestionada.

Tipo de conexión de cliente no gestionado no XA o XA

Cuando se utiliza un tipo de conexión de cliente no gestionado, puede especificar la ubicación de la tabla CCDT de dos maneras:

- Utilizando las variables de entorno MQCHLLIB para especificar el directorio donde reside la tabla, y MQCHLTAB para especificar el nombre de archivo de la tabla.
- Mediante el archivo de configuración de cliente. En la stanza CHANNELS, utilice los atributos **ChannelDefinitionDirectory** para especificar el directorio donde se encuentra la tabla y **ChannelDefinitionFile** para especificar el nombre de archivo.

Si se especifica la ubicación en el archivo de configuración de cliente y también mediante las variables de entorno, las variables de entorno tienen prioridad. Puede utilizar esta característica para especificar una ubicación estándar en el archivo de configuración del cliente, y alterarla temporalmente mediante las variable de entorno, cuando sea necesario.

Tipo de conexión de cliente gestionado

Cuando se utiliza un tipo de conexión de cliente gestionado, puede especificar la ubicación de la tabla CCDT de tres maneras:

- Mediante el archivo de configuración de aplicación de .NET. En la sección CHANNELS, utilice las claves **ChannelDefinitionDirectory** para especificar el directorio donde se encuentra la tabla y **ChannelDefinitionFile** para especificar el nombre de archivo.
- Utilizando las variables de entorno MQCHLLIB para especificar el directorio donde reside la tabla, y MQCHLTAB para especificar el nombre de archivo de la tabla.
- Mediante el archivo de configuración de cliente. En la stanza CHANNELS, utilice los atributos **ChannelDefinitionDirectory** para especificar el directorio donde se encuentra la tabla y **ChannelDefinitionFile** para especificar el nombre de archivo.

Si la ubicación se especifica en varias de estas maneras, las variables de entorno tienen prioridad sobre el archivo de configuración del cliente, y el archivo de configuración de aplicación de .NET tiene prioridad sobre los otros dos métodos. Puede utilizar esta característica para especificar una ubicación estándar en el archivo de configuración de cliente y alterarla temporalmente mediante variables de entorno o el archivo de configuración de aplicación cuando sea necesario.

Antes de IBM MQ 9.3.0, existe una diferencia de comportamiento entre el cliente .NET gestionado y los clientes IBM MQ Java y C cuando se utiliza una CCDT con agrupación de gestores de colas. Cuando un archivo CCDT contiene un grupo de gestores de colas de tres gestores de colas y tres CLNTCONNs explícitos a los mismos tres gestores de colas y la aplicación proporciona "*" como gestor de colas, los clientes C y Java devuelven MQRC_Q_MGR_NAME_ERROR. Sin embargo, el cliente .NET gestionado utiliza el primer CLNTCONN disponible y, si no hay ninguno disponible, utiliza el gestor de colas agrupado CLNTCONN.

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, el cliente .NET se comporta de la misma forma que los clientes C y Java y devuelve MQRC_Q_MGR_NAME_ERROR cuando se utiliza una CCDT con agrupación de gestores de colas.

Cómo determina una aplicación .NET qué definición de canal usar

En el entorno de cliente de IBM MQ .NET, la definición de canal que se va a utilizar se puede especificar de varias formas. Pueden existir varias especificaciones de la definición de canal. Una aplicación deduce la definición de canal a partir de una o más fuentes.

Si existe más de una definición de canal, la que se va a usar se selecciona en el orden de prioridad siguiente:

1. Propiedades especificadas en el constructor de MQQueueManager, de forma explícita o incluyendo *MQC.CHANNEL_PROPERTY* en la tabla hash de propiedades.
2. La propiedad *MQC.CHANNEL_PROPERTY* en la tabla hash MQEnvironment.properties.
3. La propiedad *Channel* en MQEnvironment.
4. El archivo de configuración de aplicación de .NET, nombre de sección CHANNELS, clave ServerConnectionParms (se aplica solo a conexiones gestionadas)
5. La variable de entorno *MQSERVER*.
6. El archivo de configuración de cliente, stanza CHANNELS, atributo ServerConnectionParms.
7. La tabla de definiciones de canal de cliente (CCDT). La ubicación de CCDT se especifica en el archivo de configuración de aplicación de .NET (solo se aplica a conexiones gestionadas)
8. La tabla de definiciones de canal de cliente (CCDT). La ubicación de CCDT se especifica utilizando las variables de entorno *MQCHLIB* y *MQCHLTAB*
9. La tabla de definiciones de canal de cliente (CCDT). La ubicación de la CCDT se especifica utilizando el archivo de configuración del cliente.

En los elementos 1-3, la definición de canal se crea campo por campo a partir de los valores proporcionados por la aplicación. Estos parámetros se pueden proporcionar utilizando interfaces diferentes y pueden existir varios valores por cada uno. Los valores de campo se añaden a la definición de canal conforme al orden de prioridad dado:

1. El valor de *connName* en el constructor de MQQueueManager.
2. Los valores de las propiedades de la tabla hash MQQueueManager.properties.
3. Los valores de las propiedades de la tabla hash MQEnvironment.properties.
4. Los valores establecidos como campos de MQEnvironment (por ejemplo, MQEnvironment.Hostname, MQEnvironment.Port).

En los elementos 4-6, se proporciona como valor la definición de canal completa. Los campos no especificados de la definición de canal toman los valores predeterminados. No se fusionan valores de otros métodos de definición de canal y sus campos con estas especificaciones.

En los elementos 7-9, se toma de la CCDT la definición de canal entera. Los campos que no se han especificado explícitamente al definirse el canal toman los valores predeterminados del sistema. No se fusionan valores de otros métodos de definición de canal y sus campos con estas especificaciones.

Utilización de salidas de canal en IBM MQ .NET

Si utiliza enlaces de cliente, puede utilizar salidas de canal como para cualquier otra conexión de cliente. Si utiliza enlaces gestionados, debe escribir un programa de salida que implemente una interfaz adecuada.

Enlaces de cliente

Si utiliza enlaces de cliente, puede utilizar salidas de canal tal como se describe en Salidas de canal. No puede utilizar salidas de canal escritas para enlaces gestionados.

Enlaces gestionados

Si utiliza una conexión gestionada, para implementar una salida, defina una nueva clase de .NET que implemente la interfaz adecuada. Están definidas tres interfaces de salida en el paquete de IBM MQ:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Nota: Las salidas de usuario escritas utilizando estas interfaces no están soportadas como salidas de canal en el entorno no gestionado.

El ejemplo siguiente define una clase que implementa las tres interfaces:

```
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
  }

}
```

A cada salida se le pasan las instancias de objeto MQChannelExit y MQChannelDefinition. Estos objetos representan las estructuras MQCXP y MQCD definidas en la interfaz de procedimientos.

Los datos que deben ser enviados por una salida de emisión, y los datos recibidos en una salida de seguridad o de recepción se especifican utilizando los parámetros de la salida.

Al inicio de la ejecución de la salida, los datos situados en el desplazamiento *dataOffset* con una longitud *dataLength* dentro de la matriz de bytes *dataBuffer* son los datos que serán enviados por una salida de emisión, y los datos que se recibirán en una salida de seguridad o recepción. El parámetro *dataMaxLength* proporciona la longitud máxima (desde *dataOffset*) disponible para la salida en *dataBuffer*. Nota: para una salida de seguridad, dataBuffer puede ser nulo si es la primera vez que invoca la salida o el interlocutor decidió no enviar ningún dato.

Al finalizar la ejecución de la salida, el valor de *dataOffset* y *dataLength* se deben establecer para que apunten a los valores de desplazamiento y longitud contenidos en la matriz de bytes devuelta, valores que deben entonces ser utilizados por las clases de .NET. Para una salida de emisión, esto indica los datos que la salida debe enviar, y para una salida de seguridad o de recepción, los datos que se deben interpretar. Normalmente, la salida debe devolver una matriz de bytes; las excepciones son una salida de seguridad que podría elegir no enviar datos, y cualquier salida invocada con las razones INIT o TERM. Por lo tanto, la forma más sencilla de salida que se puede escribir es una que no hace nada más que devolver dataBuffer:

El cuerpo de salida más simple es:

```
{
  return dataBuffer;
}
```

La clase MQChannelDefinition

El ID de usuario y la contraseña que se especifican con la aplicación cliente .NET gestionada se establecen en la clase MQChannelDefinition de IBM MQ .NET que se pasa a la salida de seguridad del cliente. La salida de seguridad copia el ID de usuario y la contraseña en los campos MQCD.RemoteUserIdentifier y MQCD.RemotePassword (consulte “Desarrollo de una salida de seguridad” en la página 999).

Especificación de salidas de canal (cliente gestionado)

Si especifica un nombre de canal y un nombre de conexión al crear el objeto MQQueueManager (en el constructor MQEnvironment o en el constructor MQQueueManager), puede especificar las salidas de canal de dos maneras.

En orden de prioridad, son:

1. Paso de las propiedades de tabla hash MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY oMQC.RECEIVE_EXIT_PROPERTY en el constructor MQQueueManager.
2. Establecimiento de las propiedades MQEnvironment SecurityExit, SendExit o ReceiveExit.

Si no especifica un nombre de canal y un nombre de conexión, las salidas de canal que se van a utilizar proceden de la definición de canal recogida desde una tabla de definición de canal de cliente (CCDT). No es posible sustituir los valores almacenados en la definición de canal. Consulte la Tabla de definiciones de canal de cliente y “Utilización de una tabla de definiciones de canal de cliente con .NET” en la página 608 para obtener más información sobre las tablas de definiciones de canal.

En cada caso, la especificación adopta la forma de una serie con el formato siguiente:

```
Assembly_name(Class_name)
```

Class_name es el nombre calificado al completo, incluyendo la especificación de espacio de nombres, de una clase de .NET que implementa la interfaz IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit o IBM.WMQ.MQReceiveExit (según corresponda). *Assembly_name* es la ubicación totalmente calificada, incluida la extensión de archivo, del conjunto que alberga la clase. La longitud de la serie está limitada a 999 caracteres si se utilizan las propiedades de MQEnvironment o MQQueueManager. Sin embargo, si el nombre de salida de canal se especifica en la CCDT, se limita a 128 caracteres. Cuando sea necesario, el código de cliente de .NET carga y crea una instancia de la clase especificada analizando la especificación de serie.

Especificación de datos de usuario de salida de canal (cliente gestionado)

Las salidas de canal pueden tener datos de usuario asociados. Si se especifica un nombre de canal y un nombre de conexión al crear el objeto MQQueueManager (en el constructor de MQEnvironment o de MQQueueManager), se pueden especificar los datos de usuario de dos maneras.

En orden de prioridad, son:

1. Pasando las propiedades de tabla hash MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY o MQC.RECEIVE_USERDATA_PROPERTY en el constructor MQQueueManager.
2. Estableciendo las propiedades SecurityUserData, SendUserData o ReceiveUserData de MQEnvironment.

Si no se especifican los nombres de canal y conexión, los valores de datos de usuario de salida que se usen procederán de la definición de canal recogida de la tabla de definiciones de canal de cliente (CCDT). No es posible sustituir los valores almacenados en la definición de canal. Consulte la Tabla de

definiciones de canal de cliente y “Utilización de una tabla de definiciones de canal de cliente con .NET” en la página 608 para obtener más información sobre las tablas de definiciones de canal.

En cada caso, la especificación es una cadena limitada a 32 caracteres.

Reconexión automática del cliente en .NET

Puede reconectar automáticamente su cliente a un gestor de colas cuando se interrumpe la conexión de forma imprevista.

Un cliente se puede desconectar de un gestor de colas de forma imprevista si, por ejemplo, se detiene el gestor de colas o la red o el servidor fallan.

Sin una reconexión automática del cliente, se produce un error si falla la conexión. Puede utilizar el código de error como ayuda para volver a establecer la conexión.

El cliente que utiliza la función de reconexión automática de cliente se denomina un cliente reconectable. Para crear un cliente reconectable especifique determinadas opciones, denominadas opciones de reconexión, durante la conexión con el gestor de colas.

Si la aplicación de cliente es un cliente de IBM MQ .NET, puede optar por obtener una reconexión automática de cliente especificando un valor adecuado para CONNECT_OPTIONS_PROPERTY, cuando utiliza la clase `MQQueueManager` para crear un gestor de colas. Consulte la sección Opciones de reconexión para obtener detalles de los valores CONNECT_OPTIONS_PROPERTY.

Puede seleccionar si la aplicación de cliente siempre se conecta y reconecta a un gestor de colas con el mismo nombre, al mismo gestor de colas o a cualquier gestor de colas definido con el mismo `QMNAME` en la tabla de conexiones de cliente. Para obtener más información, consulte la sección Grupos de gestores de colas en CCDT.

Soporte de Transport Layer Security (TLS) para .NET

Las aplicaciones cliente de IBM MQ classes for .NET permiten el cifrado TLS (Transport Layer Security). El protocolo TLS proporciona a las comunicaciones seguridad en Internet y permiten a las aplicaciones cliente/servidor comunicarse de una forma que es confidencial y fiable.

Conceptos relacionados

Soporte TLS de cliente gestionado de IBM MQ.NET

Protocolos de seguridad de cifrado: TLS

Soporte de TLS para el cliente no gestionado de .NET

El soporte TLS para el cliente .NET no gestionado se basa en C MQI y IBM Global Security Kit (GSKit). La MQI de C maneja las operaciones TLS y GSKit implementa los protocolos de socket seguro TLS.

Habilitación de TLS para el cliente .NET no gestionado

TLS sólo está soportado para conexiones de cliente. Para habilitar TLS, debe especificar la CipherSpec que se debe utilizar al comunicar con el gestor de colas, y esto debe coincidir con la CipherSpec establecida en el canal de destino.

Para habilitar TLS, especifique la CipherSpec utilizando la variable de miembro estático SSLCipherSpec de MQEnvironment. El ejemplo siguiente asocia un canal SVRCONN denominado SECURE.SVRCONN.CHANNEL, que se ha configurado de forma que requiere TLS, con una CipherSpec denominada TLS_RSA_WITH_AES_128_CBC_SHA:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Consulte Especificación de CipherSpecs para una lista de CipherSpecs.

La propiedad SSLCipherSpec también se puede establecer utilizando MQC.SSL_CIPHER_SPEC_PROPERTY en la tabla hash de las propiedades de conexión.

Para conectar correctamente mediante TLS, el almacén de claves del cliente se debe configurar con la cadena de certificados raíz de entidad emisora de certificados a partir de la cual se pueda autenticar el certificado presentado por el gestor de colas. De forma similar, si SSLClientAuth en el canal SVRCONN se ha establecido en MQSSL_CLIENT_AUTH_REQUIRED, el almacén de claves del cliente debe contener un certificado personal de identificación que sea de confianza para el gestor de colas.

Utilización del nombre distinguido del gestor de colas
El gestor de colas se identifica a sí mismo utilizando un certificado TLS, que contiene un *Nombre distinguido* (DN).

Una aplicación cliente de IBM MQ .NET puede utilizar este nombre distinguido para asegurarse de que se está comunicando con el gestor de colas correcto. Se especifica un patrón de DN utilizando la variable sslPeerName de MQEnvironment. Por ejemplo, si establece:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permite que la conexión se realice correctamente sólo si el gestor de colas presenta un certificado con un nombre común que empieza por QMGR., y al menos dos nombres de unidad organizativa, el primero de los cuales debe ser IBM y el segundo WEBSPHERE.

La propiedad SSLPeerName también se puede establecer utilizando MQC.SSL_PEER_NAME_PROPERTY en la tabla hash de las propiedades de conexión. Para obtener más información sobre nombres distinguidos y las reglas para establecer nombres del mismo nivel (peer names), consulte Protección IBM MQ.

Si se establece SSLPeerName, las conexiones sólo se realizan satisfactoriamente si se ese parámetro está establecido en un patrón válido y el gestor de colas presenta el certificado correspondiente.

Manejo de errores al utilizar TLS

IBM MQ classes for .NET puede emitir los códigos de razón siguientes cuando se conecta a un gestor de colas utilizando TLS:

MQRC_SSL_NOT_ALLOWED
Se ha establecido la propiedad SSLCipherSpec, pero se ha utilizado la conexión de enlaces. Sólo la conexión de cliente permite utilizar TLS.

MQRC_SSL_PEER_NAME_MISMATCH
El patrón de nombre distinguido especificado en la propiedad SSLPeerName no coincide con el nombre distinguido presentado por el gestor de colas.

MQRC_SSL_PEER_NAME_ERROR
El patrón de nombre distinguido especificado en la propiedad SSLPeerName no es válido.

V 9.3.0 V 9.3.0 **MQRC_KEY_REPOSITORY_ERROR**
La ubicación del repositorio de claves no se ha especificado, no es válida o no se puede acceder a ella.

Soporte de TLS para el cliente gestionado .NET

El cliente .NET gestionado utiliza las bibliotecas de Microsoft .NET Framework para implementar protocolos de socket seguro TLS. La clase System.Net.SecuritySslStream de Microsoft opera como una corriente sobre sockets TCP conectados y envía y recibe datos a través de esa conexión de socket.

El nivel mínimo necesario de .NET Framework es .NET Framework v3.5. El nivel de soporte de algoritmo de cifrado se basa en el nivel .NET Framework que utiliza la aplicación:

- Para las aplicaciones basadas en .NET Framework niveles 3.5 y 4.0, los protocolos de socket seguro disponibles son SSL 3.0 y TSL 1.0.
- Para las aplicaciones basadas en .NET Framework nivel 4.5, los protocolos de socket seguro disponibles son SSL 3.0, TLS 1.1 y TLS 1.2.

Es posible que tenga que mover las aplicaciones que esperan un soporte de protocolo TLS más alto a una versión posterior de la infraestructura tal como se define para el soporte de seguridad de Microsoft en .NET Framework.

Las características principales del soporte de TLS para el cliente .NET gestionado son las siguientes:

Soporte de protocolo TLS
El soporte de TLS para el cliente gestionado .NET se define mediante la clase SSLStream de .NET, y depende del nivel de .NET Framework que la aplicación esté utilizando. Para obtener más información, consulte el apartado "Soporte de protocolo TLS para el cliente .NET gestionado" en la página 615.

Soporte de CipherSpec
Los valores de TLS para el cliente gestionado .NET son los mismos que los utilizados para las corrientes TLS de Microsoft.NET. Para obtener más información, consulte los temas "Soporte de CipherSpec para el cliente .NET gestionado" en la página 616 y "Correlaciones de CipherSpec para el cliente .NET gestionado" en la página 617.

Repositorios de claves
El repositorio de claves en el extremo cliente es un almacén de claves de Windows. El repositorio en el extremo servidor es un repositorio de tipo Cryptographic Message Syntax (CMS). Para obtener más información, consulte el apartado "Repositorios de claves para el cliente .NET gestionado" en la página 619.

Certificados
Puede utilizar certificados TLS autofirmados para implementar la autenticación mutua entre un cliente y un gestor de colas. Para obtener más información, consulte el apartado "Utilización de certificados para el cliente gestionado .NET." en la página 620.

SSLPEERNAME
En .NET, las aplicaciones pueden utilizar el atributo opcional SSLPEERNAME para especificar un patrón de Nombre distinguido (DN). Para obtener más información, consulte el apartado "SSLPEERNAME" en la página 620.

Conformidad con FIPS
La habilitación programática de FIPS no está soportada por la biblioteca de seguridad de Microsoft.NET. La habilitación de FIPS está controlada por el valor Política de grupo de Windows.

Conformidad con NSA Suite B
IBM MQ implementa RFC 6460. La implementación de Microsoft.NET para NSA suite B es 5430. Esto está soportado en .NET Framework 3.5 y versiones posteriores.

Restablecimiento o renegociación de claves secretas
Aunque la clase SSLStream no da soporte al restablecimiento o renegociación de claves secretas, por coherencia con otros clientes IBM MQ , el cliente gestionado de .NET permite que las aplicaciones establezcan `SSLKeyResetCount`. Para obtener más información, consulte el apartado "Restablecimiento o renegociación de claves secretas para el cliente .NET gestionado" en la página 621.

Comprobación de revocación
La clase SSLStream permite la comprobación de la revocación de certificados, que se realiza automáticamente mediante el motor de encadenamiento de certificados. Para obtener más información, consulte el apartado "Comprobación de revocación" en la página 621.

Soporte de salida de seguridad de IBM MQ
La clase `SSLStream` proporciona soporte limitado para salidas de seguridad de IBM MQ . Es posible consultar certificados locales y remotos para obtener `SSLPeerNamePtr`(DN de sujeto) y `SSLRemCertIssNamePtr` (DN de emisor), ya que esto está soportado en Microsoft.NET. Pero no existe soporte para obtener atributos tales como DNQ, UNSTRUCTUREDNAME y UNSTRUCTUREDADDRESS, por lo que estos valores no se pueden recuperar utilizando las salidas.

Soporte de hardware de cifrado
El hardware de cifrado no está soportado para el cliente gestionado .NET.

Soporte para TLS1.3 en clientes IBM MQ .NET y XMS .NET gestionados

V 9.3.2

A partir de IBM MQ 9.3.2, los clientes IBM MQ .NET y XMS .NET dan soporte a TLS1.3 siempre que el sistema operativo dé soporte a TLS1.3.

El cliente .NET gestionado utiliza las bibliotecas de Microsoft .NET Framework para implementar protocolos de socket seguro TLS. La clase Microsoft `System.Net.SecuritySslStream` funciona como una corriente a través de sockets TCP conectados y envía y recibe datos a través de esa conexión de socket.

En Windows, .NET utiliza SCHANNEL, y en Linux .NET utiliza OpenSSL para la comunicación SSL.

Windows Para aplicaciones cliente de IBM MQ .NET que se ejecutan en Windows

Microsoft había anunciado que Windows 11 y Windows Server 2022 dan soporte a los cifrados TSL1.3 de forma predeterminada.

Las suites de cifrado TLS_AES_128_GCM_SHA256 y TLS_AES_256_GCM_SHA384 están habilitadas de forma predeterminada en ambas versiones de Windows.

Atención:

- TLS_CHACHA20_POLY1305_SHA256 La suite de cifrado no está habilitada de forma predeterminada, pero está soportada.
- Para un cliente de IBM MQ .NET con TLS1.3 habilitado, para conectarse correctamente a un gestor de colas, IBM Global Security Kit (GSKit) versión 8.0.55.29 es la versión mínima necesaria en el lado del gestor de colas.

Linux Para aplicaciones cliente de IBM MQ .NET que se ejecutan en Linux

Puesto que .NET utiliza OpenSSL en Linux para la comunicación SSL, para utilizar TLS1.3, OpenSSL v1.1.1 es el requisito mínimo.

Además, como .NET utiliza OpenSSL en Linux, todos los cifrados soportados por OpenSSL también deberían funcionar para .NET .

OpenSSL da soporte a las siguientes CipherSpecs para TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Conceptos relacionados

"Correlaciones de CipherSpec para el cliente .NET gestionado" en la página 617
La interfaz IBM MQ.NET mantiene una tabla de correlaciones de IBM MQ con Microsoft.NET que se utiliza para determinar la versión del protocolo TLS que necesita utilizar el cliente gestionado para establecer una conexión segura con un gestor de colas.

Soporte de protocolo TLS para el cliente .NET gestionado
El soporte de TLS de IBM MQ.NET se basa en la clase SSLStream de .NET.

Nota: El soporte de protocolo TLS para el cliente .NET gestionado depende del nivel de .NET Framework que utiliza la aplicación. Para obtener más información, consulte "Soporte de TLS para el cliente gestionado .NET" en la página 613.

Para que la clase SSLStream de Microsoft.NET inicialice TLS y realice un reconocimiento con el gestor de colas, uno de los parámetros necesarios que debe establecer es **SSLProtocol**, donde debe especificar el número de versión de TLS, que debe ser uno de los valores siguientes:

- SSL3.0
- TLS1.0
- TLS1.2

El valor de este parámetro está estrechamente vinculado con la familia de protocolos a la que pertenece la CipherSpec preferida. Cuando SSLStream inicia un reconocimiento de TLS con el servidor (gestor de colas), utiliza la versión de TLS especificada en **SSLProtocol** para identificar la lista de CipherSpecs que se utilizarán para la negociación.

IBM MQ.NET no deja propiedades disponibles para que las aplicaciones las utilicen para establecer este valor. En su lugar, IBM MQ utiliza una tabla de correlación para correlacionar internamente la CipherSpec establecida la familia de protocolos e identifica la versión de SSLProtocol que se va a utilizar. Esta tabla muestra la correlación de cada una de las CipherSpec soportadas entre Microsoft.NET y IBM MQ y la versión de protocolo a la que pertenecen. Para obtener más información, consulte "Correlaciones de CipherSpec para el cliente .NET gestionado" en la página 617.

Soporte de CipherSpec para el cliente .NET gestionado
Los valores de CipherSpec de una aplicación se utilizan durante el protocolo de conocimiento (handshake) con el servidor.

Los clientes de IBM MQ le permiten establecer un valor de CipherSpec que se utiliza durante el conocimiento con el gestor de colas. Los clientes de IBM MQ deben establecer una CipherSpec válida para que se establezca una conexión segura, preferiblemente la CipherSpec especificada en la política de grupo de Windows. Si se deja este campo en blanco, el canal estará en texto plano sin ninguna seguridad en los sockets.

En el cliente IBM MQ.NET gestionado, los parámetros de configuración de TLS son para la clase SSLStream de Microsoft.NET. En SSLStream, una CipherSpec, o lista de preferencias de CipherSpec, solo se puede configurar en la política de grupo Windows, que es una configuración a nivel de sistema. Después SSLStream usa la CipherSpec o lista de preferencias especificadas durante el conocimiento con el servidor. En el caso de otros clientes IBM MQ, la propiedad CipherSpec se puede establecer en la aplicación en la definición de canal de IBM MQ y se utiliza el mismo valor para la negociación TLS. Como resultado de esta restricción, el conocimiento TLS podría negociar cualquier CipherSpec soportada, independientemente de lo que se ha especificado en la configuración del canal IBM MQ. Por tanto, es probable que esto resulte en un error AMQ9631 del gestor de colas. Para evitar este error, establezca la misma CipherSpec que la que se ha establecido en la aplicación como configuración de TLS en la política de grupo Windows.

El nuevo código cliente de TLS de IBM MQ.NET solo comprueba que se ha negociado la versión de protocolo correcta. La versión del protocolo TLS se deriva de la CipherSpec configurada por la aplicación y se utiliza en el conocimiento TLS con el servidor (gestor de colas). Por lo tanto, por diseño es necesario establecer la CipherSpec en la aplicación cliente gestionada de IBM MQ.NET. Si la CipherSpec establecida por el cliente IBM MQ es distinta de la de los protocolos SSL 3.0, TLS 1.0 y TLS 1.2 , el cliente IBM MQ gestionado .NET negociaría de forma predeterminada con cualquiera de los cifrados de los protocolos SSL 3.0 o TLS 1.0 y no notificaría un error.

Nota: Si el valor de CipherSpec proporcionado por la aplicación no es una CipherSpec conocida por IBM MQ, el cliente IBM MQ gestionado .NET no lo tiene en cuenta y negocia la conexión basada en la política de grupo del sistema Windows .

Definición de una CipherSpec

Hay tres formas de establecer una CipherSpec:

clase MQEnvironment .NET
En el ejemplo siguiente se muestra cómo establecer una CipherSpec con la clase MQEnvironment.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

Propiedad CipherSpec de TLS

En el ejemplo siguiente se muestra cómo establecer una CipherSpec añadiendo un parámetro de tabla hash en el constructor de MQQueueManager.

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Política de grupo Windows

Cuando se configura una lista de suites de cifrado a través de la consola de gestión de políticas de grupo de Windows , la definición de canal SVRCONN debe especificar una CipherSpeccoincidente. Una CipherSpec coincidente podría ser un valor genérico como "ANY_TLS12_OR_HIGHER", o un valor específico que se correlaciona con la suite de cifrado más alta que se negociaría desde la lista ordenada. Se recomienda el uso de valores CipherSpec genéricos para su uso con clientes .NET, ya que evita la necesidad de cambiar la configuración de SVRCONN CipherSpec si el orden de la lista de clientes cambia.

Uso de la CCDT

IBM MQ.NET solo da soporte a las tablas de definiciones de canal cliente (archivos .TAB) que están en un sistema local. Los archivos CCDT existentes que tienen un valor de CipherSpec establecido se pueden utilizar para las conexiones de IBM MQ.NET. Sin embargo, el valor de CipherSpec establecido en el canal de conexión de cliente determina la versión de protocolo TLS y, también, debe coincidir con la CipherSpec establecida en la política de grupo Windows.

Conceptos relacionados

“Configuración del entorno de IBM MQ” en la página 603
Antes de utilizar la conexión de cliente para conectar con gestor de colas, debe configurar el entorno de IBM MQ.

“Soporte de TLS para el cliente gestionado .NET” en la página 613
El cliente .NET gestionado utiliza las bibliotecas de Microsoft .NET Framework para implementar protocolos de socket seguro TLS. La clase System.Net.SecuritySslStream de Microsoft opera como una corriente sobre sockets TCP conectados y envía y recibe datos a través de esa conexión de socket.

Tareas relacionadas

Especificación de CipherSpecs

Referencia relacionada

clase MQEnvironment .NET

Correlaciones de CipherSpec para el cliente .NET gestionado

La interfaz IBM MQ.NET mantiene una tabla de correlaciones de IBM MQ con Microsoft.NET que se utiliza para determinar la versión del protocolo TLS que necesita utilizar el cliente gestionado para establecer una conexión segura con un gestor de colas.

Si se especifica una CipherSpec en el canal SVRCONN, una vez que haya concluido el protocolo de reconocimiento de TLS, el gestor de colas compara esa CipherSpec con la CipherSpec negociada que la aplicación cliente está utilizando. Si el gestor de colas no puede encontrar una CipherSpeccoincidente, la comunicación falla con el error AMQ9631.

La interfaz de IBM MQ.NET mantiene una tabla de correlación de CipherSpec de IBM MQ en Microsoft.NET. Esta tabla se utiliza para determinar la versión del protocolo TLS que el cliente desea utilizar para establecer una conexión de socket segura con el gestor de colas. Basándose en el valor SSLCipherSpec, la versión de SSLProtocol puede ser TLS 1.0, o TLS 1.2, en función de la versión de la infraestructura de Microsoft.NET que esté utilizando.

Asegúrese de que proporciona el valor SSLCipherSpec correcto, ya que un valor incorrecto podría provocar que se utilicen los protocolos SSL 3.0 o TLS 1.0.

Tabla 78. Tabla de correlación de IBM MQ y Microsoft.NET

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versión de TLS**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2

Tabla 78. Tabla de correlación de IBM MQ y Microsoft.NET (continuación)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versión de TLS**
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.3.2 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.3.2 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

Notas:

1. Deprecated La especificación de cifrado TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.

Conceptos relacionados

"Soporte de TLS para el cliente gestionado .NET" en la página 613
El cliente .NET gestionado utiliza las bibliotecas de Microsoft .NET Framework para implementar protocolos de socket seguro TLS. La clase System.Net.SecuritySslStream de Microsoft opera como una corriente sobre sockets TCP conectados y envía y recibe datos a través de esa conexión de socket.

Repositorios de claves para el cliente .NET gestionado
El repositorio de claves utilizado por los clientes .NET gestionados es el almacén de claves Windows . Los certificados y las claves privadas deben estar disponibles en el almacén de claves del usuario o del sistema para que la aplicación cliente pueda utilizarlos para la identidad y la confianza durante un reconocimiento TLS.

Lado cliente

En la aplicación, se puede establecer cualquiera de los valores siguientes para el depósito de claves:

- "*USER": IBM MQ.NET accede al almacén de certificados del usuario actual para recuperar los certificados de cliente.
- "*SYSTEM": IBM MQ.NET accede a la cuenta del sistema local para recuperar los certificados.

Los certificados del cliente tienen que almacenarse en el almacén de certificados de la cuenta de usuario o de sistema. Todos los certificados de servidor (CA) tienen que almacenarse en el directorio raíz del almacén de certificados.

Nota: Puede almacenar más de un certificado en un único archivo en los formatos siguientes:

- Intercambio de información personal-PKCS #12 (.PFX, .P12)

- Estándar de sintaxis de mensajes criptográficos-Certificados PKCS #7 (.P7B)
- Almacén de certificados serializados de Microsoft (.SST)

Utilización de certificados para el cliente gestionado .NET.
Para los certificados de cliente, el cliente IBM MQ gestionado .NET accede al almacén de claves Windows y carga todos los certificados del cliente que coinciden con la etiqueta de certificado o con la serie.

Al seleccionar un certificado para utilizar, el cliente IBM MQ gestionado .NET siempre utiliza el primer certificado coincidente para el reconocimiento TLS de SSLStream.

Certificados coincidentes por etiqueta de certificado

Si establece la etiqueta de certificado, el cliente IBM MQ gestionado .NET busca el almacén de certificados de Windows con el nombre de etiqueta proporcionado para identificar el certificado de cliente. Carga todos los certificados coincidentes y utiliza el primer certificado de la lista. Hay dos opciones para establecer la etiqueta de certificado:

- La etiqueta de certificado se puede establecer en la clase MQEnvironment que accede a MQEnvironment.CertificateLabel.
- La etiqueta de certificado también se puede establecer en las propiedades de la tabla hash, proporcionada como un parámetro de entrada con el constructor MQQueueManager como se muestra en el ejemplo siguiente.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

El nombre ("CertificateLabel") y el valor distinguen entre mayúsculas y minúsculas.

Certificados coincidentes por serie

Si la etiqueta de certificado no está establecida, se busca y se utiliza el certificado que coincide con la serie `"ibmwebspheremq"` y el usuario conectado actualmente (en mayúsculas).

Tareas relacionadas
Conexión de un cliente a un gestor de colas de forma segura
Referencia relacionada
clase MQEnvironment .NET

SSLPEERNAME
El atributo SSLPEERNAME se utiliza para comprobar el nombre distinguido (DN) del certificado del gestor de colas de iguales.

En IBM MQ.NET, las aplicaciones pueden utilizar SSLPEERNAME para especificar un patrón de nombre distinguido tal como se muestra en el ejemplo siguiente.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Para los demás clientes de IBM MQ, SSLPEERNAME es un parámetro opcional.

Si no se establece el valor de SSLPEERNAME, el cliente gestionado de IBM MQ.NET no realiza ninguna validación de certificado remoto (servidor) y el cliente gestionado solo acepta el certificado remoto (/server) tal cual.

La forma en la que se establece SSLPEERNAME depende de cuál de las ofertas de pila de IBM MQ esté utilizando.

IBM MQ classes for .NET
Hay tres opciones, como se muestra a continuación.

1. Establecer MQEnvironment.SSLPeerName en la clase MQEnvironment.

2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Utilice el constructor del gestor de colas `MQQueueManager (String queueManagerName, Hashtable properties)`. Proporcione SSLPEERNAME en `Hashtable properties` como en la opción 2.

XMS .NET

Establezca el nombre de igual SSL en la fábrica de conexiones:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

Incluya SslPeerName como un campo separado por punto y coma en el URI.

Referencia relacionada

clase MQEnvironment .NET

Restablecimiento o renegociación de claves secretas para el cliente .NET gestionado

La clase SSLStream no da soporte restablecimiento/renegociación de claves secretas. Sin embargo, para ser coherente con otros clientes IBM MQ , el cliente IBM MQ gestionado .NET permite que las aplicaciones establezcan **SSLKeyResetCount**.

Cuando se alcanza el límite, IBM MQ.NET se desconecta del gestor de colas y se notifica a la aplicación de ello como una excepción con MQRC_CONNECTION_BROKEN como código de razón. Las aplicaciones pueden elegir manejar la excepción y volver a establecer las conexiones o habilitar la opción MQCNO_RECONNECT para IBM MQ.NET para volver a conectarse automáticamente al gestor de colas.

La habilitación del recurso de reconexión de cliente automática significa que, cuando se alcanza el recuento de restablecimiento de claves, todas las conexiones existentes se descartan y el cliente de IBM MQ.NET vuelve a crear todas las conexiones a partir de cero. Para obtener más información sobre la reconexión automática de cliente, consulte Reconexión automática de cliente.

Conceptos relacionados

Restablecimiento de claves secretas SSL y TLS

Comprobación de revocación

La clase SSLStream da soporte a la comprobación de revocación de certificados.

El motor de cadena de certificados realiza la comprobación de revocación de forma automática. Esto se aplica al protocolo OCSP (Online Certificate Status Protocol) y a las listas CRL (Certificate Revocation Lists). La clase SSLStream utiliza la revocación de certificados que solo utiliza el servidor que se ha especificado en el certificado, esto es, el servidor que dicta el propio certificado. Las extensiones HTTP CDP y las solicitudes OCSP HTTP pueden comunicarse por proxy con el servidor proxy HTTP.

El modo en que establece la comprobación de revocación depende de las ofertas de la pila de IBM MQ que esté utilizando.

IBM MQ.NET

La comprobación de revocación se puede establecer accediendo a la propiedad **MQEnvironment.SSLCertRevocationCheck** del archivo de clases `MQEnvironment.cs`.

XMS .NET

La comprobación se puede establecer en el contexto de la propiedad de la fábrica de conexiones, como se muestra en el ejemplo siguiente.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF

La comprobación de revocación se puede establecer en el URI utilizando el convenio de nombres siguientes.

```
"SslCertRevocationCheck=true"
```

Configuración de TLS en IBM MQ .NET gestionado
La configuración de un IBM MQ .NET consiste en crear el certificado de firmante y luego configurar el lado del servidor, el lado del cliente y la aplicación.

Acerca de esta tarea

Para configurar TLS, primero hay que crear los correspondientes certificados de firmante. Los certificados de firmante pueden ser autofirmados o proporcionados por una entidad emisora de certificados. Aunque los certificados autofirmados se pueden utilizar en un sistema de desarrollo, de prueba o de preproducción, no los use en un sistema de producción. En un sistema de producción, utilice los certificados que haya obtenido de una entidad emisora de certificados (CA) externa de confianza.

Procedimiento

1. Cree los certificados de firmante.
   a) Para crear certificados autofirmados, utilice una de las herramientas siguientes proporcionadas con IBM MQ :

      Utilice la GUI de **strmqikm** o utilice **runmqckm** o **runmqakm** desde la línea de mandatos. Para obtener más información sobre cómo utilizar estas herramientas, consulte Utilización de **runmqckm**, **runmqakm**y **strmqikm** para gestionar certificados digitales.
   b) Para obtener de una entidad emisora de certificados (CA) los certificados para el gestor de colas y para los clientes, siga las instrucciones de la sección Obtención de certificados personales de una entidad emisora de certificados.
2. Configure el lado del servidor.
   a) Configure TLS en el gestor de colas, utilizando IBM Global Security Kit (GSKit), tal como se describe en Conexión de un cliente a un gestor de colas de forma segura.
   b) Establezca los atributos de TLS del canal SVRCONN:

      - Establezca **SSLCAUTH** a `"REQUIRED/OPCIONAL"`.
      - Establezca **SSLCIPH** a una CipherSpec adecuada.

      Para obtener más información, consulte “Habilitación de TLS para el cliente .NET no gestionado” en la página 612.
3. Configure el lado del cliente.
   a) Importe los certificados de cliente en el almacén de certificados de Windows (bajo la cuenta de Usuario/Sistema).

      IBM MQ .NET accede a los certificados de cliente desde el almacén de certificados de Windows, por lo tanto, debe importar los certificados en el almacén de certificados de Windows para establecer una conexión de socket seguro con IBM MQ . Para obtener más información sobre cómo acceder al almacén de claves de Windows e importar los certificados del lado del cliente, consulte Importar o exportar certificados y claves privadas.
   b) Proporcione la CertificateLabel (etiqueta de certificado) tal y como se describe en Conectar un cliente con un gestor de colas de forma segura.
   c) Si es necesario, edite la política de grupo de Windows para establecer la CipherSpec y, a continuación, para que las actualizaciones de la política de grupo de Windows entren en vigor, reinicie el sistema.
4. Configure el programa de aplicación.
   a) Establezca el valor MQEnvironment o SSLCipherSpec para indicar que la conexión está protegida.

      El valor que especifique se utiliza para identificar el protocolo que se utiliza (TLS). La CipherSpec definida debería ser una de las CipherSpec de la versión de SSLProtocol soportada y preferiblemente será la misma que se especifique en la política de grupo de Windows. (La versión de SSLProtocol soportada depende de la infraestructura de .NET utilizada. La versión de SSLProtocol puede ser TLS 1.0, o TLS 1.2, en función de la versión de la infraestructura de Microsoft .NET que esté utilizando).

Nota: Si el valor de CipherSpec proporcionado por la aplicación no es una CipherSpec conocida por IBM MQ, el cliente IBM MQ gestionado .NET no lo tiene en cuenta y negocia la conexión basada en la política de grupo del sistema Windows .

b) Establezca la propiedad SSLKeyRepository a "*SYSTEM" o a "*USER".

c) Opcional: Establezca SSLPEERNAME al nombre distinguido (DN) del certificado de servidor.

d) Proporcione la CertificateLabel (etiqueta de certificado) tal y como se describe en Conectar un cliente con un gestor de colas de forma segura.

e) Establezca los parámetros opcionales adicionales que necesite, por ejemplo, KeyResetCount o CertificationRevocationCheck, y habilite FIPS.

Ejemplos de cómo configurar el protocolo TLS y el repositorio de claves TLS

En el caso de .NET base, se pueden configurar el protocolo TLS y el repositorio de claves TLS a través de la clase MQEnvironment, tal como se muestra en el ejemplo siguiente:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

De forma alternativa, se pueden configurar el protocolo TLS y el repositorio de claves TLS proporcionando una tabla hash en el constructor de MQQueueManager, tal como se muestra en el ejemplo siguiente.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Qué hacer a continuación

Para obtener más información sobre cómo empezar a desarrollar aplicaciones TLS gestionadas en IBM MQ .NET, consulte “Desarrollo de una aplicación sencilla” en la página 623.

Referencia relacionada

clase MQEnvironment .NET

KeyResetCount (MQLONG)

Federal Information Processing Standards (FIPS) para AIX, Linux, and Windows

Desarrollo de una aplicación sencilla

Sugerencias para desarrollar una aplicación IBM MQ .NET TLS gestionada, incluyendo ejemplos de configuración de propiedades SSL de fábricas de conexiones, creación de instancias, conexiones, sesiones y destinos de gestor de colas, y envío de un mensaje de texto.

Antes de empezar

En primer lugar, hay que configurar TLS para IBM MQ.NET gestionado, tal y como se describe en “Configuración de TLS en IBM MQ .NET gestionado” en la página 622.

Para la configuración del programa de aplicación en .NET básico, establezca las propiedades SSL utilizando la clase MQEnvironment o pasando una tabla hash al constructor de MQQueueManager.

Para la configuración del programa de aplicación en XMS .NET, establezca las propiedades SSL en el contexto de propiedad de las fábricas de conexiones.

Procedimiento

1. Establezca las propiedades SSL de las fábricas de conexiones tal como se muestra en los ejemplos siguientes.

Ejemplo para IBM MQ.NET

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
properties.Add("CertificateLabel", "ibmwebspheremq");
MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
```

Ejemplo para XMS .NET

```
cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

2. Cree las conexiones, la sesión, el destino y la instancia del gestor de colas tal como se muestra en los ejemplos siguientes.

Ejemplo para MQ .NET

```
queueManager = new MQQueueManager(queueManagerName, properties);
Console.WriteLine("done");

// accessing queue
Console.Write("Accessing queue " + queueName + ".. ");
queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
MQC.MQOO_FAIL_IF_QUIESCING);
Console.WriteLine("done");
```

Ejemplo para XMS .NET

```
connectionWMQ = cf.CreateConnection();
// Create session
sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

// Create destination
destination = sessionWMQ.CreateQueue(destinationName);

// Create producer
producer = sessionWMQ.CreateProducer(destination);
```

3. Envíe un mensaje tal y como se muestra en los ejemplos siguientes:

Ejemplo para MQ .NET

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

Ejemplo para XMS .NET

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. Verifique la conexión TLS.

   Compruebe el estado del canal para verificar que la conexión TLS y se ha establecido y está funcionando correctamente.

Configuración del rastreo para SSLStream
Para capturar los sucesos de rastreo y los mensajes relacionados con la clase SSLStream, debe añadir una sección de configuración para los diagnósticos de sistema en el archivo de configuración de la aplicación.

Acerca de esta tarea

Nota:

Esta tarea sólo se aplica a IBM MQ classes for .NET Framework . El archivo de configuración de la aplicación no está soportado en IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6).

Si no añade una sección de configuración para los diagnósticos del sistema al archivo de configuración de la aplicación, el cliente IBM MQ gestionado .NET no capturará los sucesos, rastreos o puntos de depuración relacionados con TLS y la clase SSLStream.

Nota: El inicio del rastreo de IBM MQ utilizando **strmqtrc** no captura todo el rastreo de TLS necesario.

Procedimiento

1. Cree un archivo de configuración de aplicación (App.Config) para el proyecto de aplicación.
2. Añada una sección de configuración de diagnósticos de sistema como se muestra en el ejemplo siguiente:

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
        <trace autoflush="true"/>
     </system.diagnostics>
```

Atención: El campo `Version` de la entrada `add name` debe ser cualquier versión del archivo .net `amqmdnet.dll` que se esté utilizando.

Tareas relacionadas

Rastreo de clientes IBM MQ classes for .NET Framework utilizando un archivo de configuración de aplicación

Aplicaciones de ejemplo para implementar TLS en .NET gestionado

Se proporcionan aplicaciones de ejemplo para mostrar la implementación de TLS para .NET gestionado en IBM MQ classes for .NET, XMS .NET y canal personalizado de IBM MQ para WCF.

En la tabla siguiente, se muestra la ubicación de las aplicaciones de ejemplo. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.

Tabla 79. Ubicación de las aplicaciones de ejemplo para implementar TLS en .NET gestionado

Oferta de pila de IBM MQ.NET	**Ubicación de los ejemplos**
.NET base	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs` *MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs`
XMS .NET	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs` *MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs`
Canal personalizado de IBM MQ personalizado para WCF	*MQ_INSTALLATION_PATH*`\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs`

Windows Utilización de .NET Monitor

.NET Monitor es una aplicación similar a un supervisor desencadenante de IBM MQ.

Importante: Consulte Características que solo se pueden utilizar con la instalación primaria en Windows para obtener información importante.

Puede crear componentes de .NET que se instancian siempre que se recibe un mensaje en una cola supervisada y que, después, procesan dicho mensaje. .NET Monitor se inicia con el comando **runmqdnm** y se para con el comando **endmqdnm**. Para obtener información detallada sobre estos comandos, consulte runmqdnm y endmqdnm.

Para utilizar .NET Monitor, escriba un componente que implemente la interfaz IMQObjectTrigger, que se define en amqmdnm.dll.

Los componentes pueden ser transaccionales o no transaccionales. Un componente transaccional tiene que heredar de System.EnterpriseServices.ServicedComponent y estar registrado como RequiresTransaction o SupportsTransaction. No se debe registrar como RequiresNew ya que .NET Monitor ya ha iniciado una transacción.

El componente recibe los objetos MQQueueManager, MQQueue y MQMessage de **runmqdnm**. También puede recibir una cadena de parámetro de usuario, si se ha especificado una, con la opción de línea de comandos *-u*, al iniciar `runmqdnm`. Tenga en cuenta que el componente recibe el contenido de un mensaje que ha llegado a la cola supervisada en un objeto MQMessage. No tiene que conectarse con el gestor de colas, abrir la cola ni obtener el propio mensaje. El componente debe procesar el mensaje según corresponda y devolver el control a .NET Monitor.

Si el componente se ha desarrollado como un componente transaccional, se registra para confirmar o retrotraer la transacción utilizando los recursos proporcionados por System.EnterpriseServices.ServicedComponent.

Puesto que el componente recibe objetos MQQueueManager y MQQueue, así como el mensaje, tiene información de contexto completa para dicho mensaje y puede, por ejemplo, abrir otra cola en el mismo gestor de colas sin tener que conectarse por separado a IBM MQ.

Windows *Ejemplos de fragmentos de código*

En este tema se incluyen dos ejemplos de componentes que obtienen un mensaje del supervisor de .NET y lo imprimen, uno mediante un proceso transaccional y otro mediante un proceso no transaccional. Un tercer ejemplo muestra las rutinas de programa de utilidad más comunes aplicables a los dos primeros ejemplos. Todos los ejemplos están en C#.

Ejemplo 1: Proceso transaccional

```
/*********************************************************************/
/* 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;
    }
  }
}
```

Ejemplo 2: Proceso no transaccional

```
/*********************************************************************/
/* 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());
      }
    }
  }
}
```

Ejemplo 3: Rutinas comunes

```
/*********************************************************************/
/* 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;
    }
  }
}
```

compilar programas IBM MQ .NET

Mandatos de ejemplo para compilar aplicaciones .NET escritas en diversos lenguajes.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para crear una aplicación C# utilizando IBM MQ classes for .NET, utilice el este mandato:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

Para crear una aplicación Visual Basic utilizando IBM MQ classes for .NET, utilice este mandato:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Para crear una aplicación C++ gestionada utilizando IBM MQ classes for .NET, utilice este mandato:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Para los demás lenguajes, consulte la documentación proporcionada por el proveedor del lenguaje.

Utilización del cliente autónomo de IBM MQ .NET

El cliente de IBM MQ .NET le ofrece la posibilidad de empaquetar y desplegar un conjunto de IBM MQ .NET sin necesidad de utilizar la instalación completa del cliente de IBM MQ en sistemas de producción para ejecutar las aplicaciones.

Antes de empezar

V 9.3.1 A partir de IBM MQ 9.3.1, la biblioteca de cliente `amqmdnetstd.dll` instalada en la ubicación predeterminada se basa en .NET 6. La biblioteca de cliente de `amqmdnetstd.dll` basada en .NET Standard se ha movido a una nueva ubicación en el paquete de instalación del cliente de IBM MQ y ahora está disponible en las ubicaciones siguientes:

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`
- Linux En Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`

LTS Para IBM MQ 9.3.0 Long Term Support, la biblioteca `amqmdnetstd.dll` está disponible en las ubicaciones siguientes:

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin`. Las aplicaciones de ejemplo se instalan en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux En Linux: *MQ_INSTALLATION_PATH*`/lib64 path`. Los ejemplos de .NET se encuentran en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

LTS Stabilized La biblioteca `amqmdnet.dll` se sigue suministrando, pero esta biblioteca se estabiliza; es decir, no se introducirán nuevas características. Para cualquiera de las características más recientes, debe migrar a la biblioteca `amqmdnetstd.dll`. Sin embargo, puede seguir utilizando la biblioteca `amqmdnet.dll` en los releases de IBM MQ 9.1 Long Term Support o Continuous Delivery.

Acerca de esta tarea

Puede crear las aplicaciones de IBM MQ .NET en una máquina en la que esté instalado el cliente IBM MQ completo y posteriormente empaquetar el conjunto de IBM MQ .NET , es decir, `amqmdnetstd.dll`, junto con la aplicación y desplegarlo en sistemas de producción.

Las aplicaciones que crea y despliega pueden ser las aplicaciones tradicionales de .NET , Servicios o aplicaciones Microsoft Azure Web/Worker

En estos despliegues, el cliente de IBM MQ .NET sólo da soporte a la modalidad gestionada de conectividad con un gestor de colas. Los enlaces de servidor y la conectividad en modalidad de cliente no gestionado no están disponibles ya que estas dos modalidades requieren una instalación completa del cliente IBM MQ. Cualquier intento de utilizar estas otras dos modalidades da como resultado una excepción de la aplicación.

Procedimiento

Cómo hacer referencia al conjunto de clientes de IBM MQ .NET en aplicaciones

- Haga referencia al conjunto de `amqmdnetstd.dll` en la aplicación de la misma forma que lo hizo para releases anteriores.

  Establezca la propiedad **CopyLocal** del conjunto `amqmdnetstd.dll` en `True` para asegurarse de que el conjunto `amqmdnetstd.dll` se copia en el directorio `bin` de la aplicación. El establecimiento de esta propiedad también ayuda a la herramienta de empaquetado de aplicaciones a empaquetar los archivos binarios necesarios para el despliegue en los sistemas de producción, así como los entornos de nube Microsoft Azure PaaS.

Adición de soporte de transacciones globales

- Asegúrese de que la aplicación desplegará la aplicación de supervisión WMQDotnetXAMonitor en la máquina junto con la propia aplicación.

  Si la aplicación utiliza la función de transacción global gestionada de IBM MQ .NET, deberá desplegar también WMQDotnetXAMonitor en la máquina, junto con la propia aplicación. Este programa de utilidad es necesario para la recuperación de cualquier transacción dudosa.

Inicio y detención del rastreo

- Sólo para IBM MQ classes for .NET Framework , para iniciar y detener el rastreo utilizando el archivo de configuración de aplicación y un archivo de configuración de rastreo específico de IBM MQ , consulte Rastreo de un cliente de IBM MQ para .NET Framework utilizando un archivo de configuración de aplicación.

  Debe utilizar el archivo de configuración de la aplicación y un archivo de configuración de rastreo específico de IBM MQ porque, puesto que no hay ninguna instalación de cliente completa de IBM MQ, las herramientas estándar que se utilizan para iniciar y detener el rastreo, **strmqtrc** y **endmqtrc**, no están disponibles.

  Notas:

  - Esta forma de generar el rastreo se aplica al cliente gestionado redistribuible de .NET , así como al cliente .NET autónomo. Consulte .NET application runtime- Windows only.
  - El archivo de configuración de la aplicación no está soportado en IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6). Para habilitar el rastreo para IBM MQ classes for .NET (bibliotecas.NET Standard y .NET 6), utilice la variable de entorno **MQDOTNET_TRACE_ON** . Consulte Rastreo de aplicaciones IBM MQ .NET utilizando variables de entorno.
- V 9.3.3

  Inicie y detenga el rastreo utilizando el archivo `mqclient.ini` y estableciendo las propiedades adecuadas de la stanza Trace.

  Consulte Rastreo de aplicaciones de IBM MQ .NET con mqclient.ini.

  A partir de IBM MQ 9.3.3, puede configurar el rastreo utilizando el archivo `mqclient.ini` y estableciendo las propiedades adecuadas de la stanza Trace. También puede habilitar e inhabilitar dinámicamente el rastreo con el archivo `mqclient.ini` .

Habilitación de la redirección de enlaces en el archivo de configuración de aplicación

- Para habilitar la referencia de enlace de tiempo de compilación del ensamblaje de IBM MQ .NET a una versión posterior del ensamblaje, añada la propiedad <dependentAssembly> al archivo de configuración de la aplicación.

  El ejemplo de fragmento de código siguiente en el archivo `app.config` redirecciona una aplicación que se ha compilado utilizando la versión de IBM MQ 8.0.0 Fix Pack 2 (`8.0.0.2`) del ensamblaje IBM MQ .NET, pero posteriormente un fixpack, IBM MQ 8.0.0 Fix Pack 3, se ha aplicado a continuación, ha actualizado el ensamblaje IBM MQ.NET a `8.0.0.3`.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Conceptos relacionados

“Instalación del IBM MQ classes for .NET” en la página 572
IBM MQ classes for .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux

Clientes redistribuibles

Tiempo de ejecución de la aplicación .NET - Solamente en Windows

Tareas relacionadas

“Utilización de la aplicación WMQDotnetXAMonitor” en la página 592

El cliente IBM MQ .NET proporciona una aplicación de Monitor XA, WmqDotnetXAMonitor, que se puede utilizar para recuperar cualquier transacción distribuida incompleta. La aplicación WmqDotnetXAMonitor establece una conexión con el gestor de colas donde las transacciones están pendientes y, a continuación, resuelve la transacción basándose en los parámetros que ha establecido.

Rastreo de aplicaciones .NET de IBM MQ

V 9.3.0 OutboundSNI propiedad

Puede establecer la propiedad **OutboundSNI** en una aplicación utilizando una propiedad o una variable de entorno.

A partir de IBM MQ 9.3.0, puede establecer MQC.OUTBOUND_SNI_PROPERTY en la aplicación, utilizando una tabla hash cuando se utiliza la clase `MQQueueManager` para conectarse al gestor de colas.

La MQC de MQC.OUTBOUND_SNI_PROPERTY toma los valores siguientes:

- MQC.OUTBOUND_SNI_CHANNEL, que se correlaciona con "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME, que se correlaciona con "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK, que se correlaciona con "*"

Además, puede establecer la propiedad **OutboundSNI** utilizando la variable de entorno MQOUTBOUND_SNI, que toma los valores siguientes:

- CHANNEL
- HOSTNAME
- *

y establezca el valor **OutboundSNI** en el archivo `App.config` , como con cualquier otra propiedad `mqclient.ini` .

Nota: El valor predeterminado de la propiedad es MQC.OUTBOUND_SNI_CHANNEL si no se establece ningún valor específico.

El orden de prioridad para establecer la propiedad **OutboundSNI** en el nodo gestionado es:

1. Propiedad de nivel de aplicación
2. Variable de entorno

Para la propiedad **OutboundSNI** en un nodo no gestionado, sólo se da soporte a `mqclient.ini` .

Las propiedades establecidas en el archivo `App.config` sólo son aplicables para aplicaciones .NET Framework .

Si proporciona un valor que no es válido a nivel de aplicación o en el archivo `App.config` , se emite el código de retorno MQRC_OUTBOUND_SNI_NOT_VALID.

Si establece una variable de entorno que no es válida, o proporciona un valor que no es válido en el archivo `mqclient.ini` , se utiliza el valor predeterminado de `CHANNEL` .

OutboundSNI y varios certificados

IBM MQ utiliza la cabecera SNI para proporcionar varias funciones de certificados. Si una aplicación se está conectando a un canal IBM MQ que está configurado para utilizar un certificado diferente a través del campo CERTLABL, la aplicación debe conectarse con un valor **OutboundSNI** de CHANNEL.

Si una aplicación con un valor **OutboundSNI** distinto de CHANNEL se conecta a un canal con una etiqueta de certificado configurada, la aplicación se rechaza con un MQRC_SSL_INITIALIZATION_ERROR y se imprime un mensaje AMQ9673 en los registros de errores del gestor de colas.

Para obtener más información sobre cómo IBM MQ proporciona varias funciones de certificados, consulte Cómo IBM MQ proporciona varias funciones de certificados .

Desarrollo de aplicaciones XMS .NET

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) proporciona una interfaz de programación de aplicaciones (API) llamada XMS que tiene el mismo conjunto de interfaces que la API Java Message Service (JMS). IBM MQ Message Service Client (XMS) for .NET contiene una implementación totalmente gestionada de XMS, que puede ser utilizada por cualquier lenguaje compatible con .NET.

Acerca de esta tarea

XMS admite:

- Mensajería punto a punto
- Mensajería de publicación/suscripción
- Entrega de mensajes síncrona
- Entrega de mensajes asíncrona

Una aplicación XMS puede intercambiar mensajes con los tipos de aplicación siguientes:

- Una aplicación XMS
- Una aplicación IBM MQ classes for JMS
- Una aplicación nativa IBM MQ
- Una aplicación JMS que está utilizando el proveedor de mensajería IBM MQ predeterminado

Una aplicación XMS se puede conectar a, y utilizar los recursos de, cualquiera de los servidores de mensajería siguientes:

Gestor de colas IBM MQ
La aplicación se puede conectar en la modalidad de enlaces o cliente.

WebSphere Application Server service integration bus
La aplicación puede utilizar una conexión TCP/IP directa, o puede utilizar HTTP sobre TCP/IP.

IBM Integration Bus
Los mensajes se transportan entre la aplicación y el intermediario utilizando WebSphere MQ Real-Time Transport. Los mensajes se pueden entregar a la aplicación utilizando WebSphere MQ Multicast Transport.

Al conectarse a un gestor de colas IBM MQ , una aplicación XMS puede utilizar WebSphere MQ Enterprise Transport para comunicarse con IBM Integration Bus. De forma alternativa, una aplicación XMS puede publicar y suscribirse a conectándose a IBM MQ.

A partir de IBM MQ 9.1.1, IBM MQ da soporte a .NET Core para aplicaciones en entornos Windows. Para obtener más información, consulte “Instalación del IBM MQ classes for XMS .NET” en la página 637.

A partir de IBM MQ 9.1.2, IBM MQ da soporte a .NET Core para aplicaciones en entornos Linux.

A partir de IBM MQ 9.1.4, las aplicaciones gestionadas de XMS .NET pueden equilibrar automáticamente las conexiones entre los gestores de colas agrupados en clústeres. Se da soporte a las bibliotecas .NET Framework y .NET Standard. Para obtener más información, consulte Acerca de los clústeres uniformes y Equilibrio automático de aplicaciones.

Tareas relacionadas
Cómo ponerse en contacto con el servicio de soporte de IBM
Resolución de problemas de XMS .NETproblems

Estilos de mensajería soportados por XMS

XMS admite los estilos de punto a punto y de publicación/suscripción de la mensajería.

Los estilos de mensajería también se denominan dominios de mensajería.

Mensajería punto a punto

Una forma común de la mensajería punto a punto utiliza la colocación en colas. En el caso más sencillo, una aplicación envía un mensaje a otra aplicación identificando, de forma implícita o explícita, una cola de destino. La sistema subyacente de mensajería y colocación en colas recibe el mensaje de la aplicación emisora y direcciona el mensaje a su cola de destino. La aplicación receptora puede recuperar el mensaje de la cola.

Si el sistema subyacente de mensajería y colocación en cola contiene IBM Integration Bus, IBM Integration Bus podría duplicar un mensaje y direccionar copias del mensaje a distintas colas. Como resultado, más de una aplicación puede recibir el mensaje. IBM Integration Bus también podría transformar un mensaje y añadirle datos.

Una característica clave de la mensajería punto a punto es que una aplicación coloca un mensaje en una cola local cuando envía un mensaje. EL sistema subyacente de mensajería y colocación en cola determina a qué cola de destino se envía el mensaje. La aplicación receptora recupera el mensaje de la cola de destino.

Mensajería de publicación/suscripción

En la mensajería de publicación/suscripción, existen dos tipos de aplicación: publicador y suscriptor.

Un *publicador* proporciona información en forma de mensajes de publicación. Cuando un publicador publica un mensaje, especifica un tema, que identifica el tema de la información incluida en el mensaje.

Un *suscriptor* es un consumidor de la información que se publica. Un suscriptor especifica los temas en los que está interesado creando suscripciones.

El sistema de publicación/suscripción recibe publicaciones de publicadores y suscripciones de suscriptores. Direcciona publicaciones a suscriptores. Un suscriptor recibe publicaciones sobre solo aquellos temas a los que está suscrito.

Una característica clave de la mensajería de publicación/suscripción es que un publicador identifica un tema cuando publica un mensaje. No identifica los suscriptores. Si se publica un mensaje sobre un tema para el cual no hay suscriptores, ninguna aplicación recibe el mensaje.

Una aplicación puede ser a la vez publicador y suscriptor.

Modelo de objeto XMS

La API XMS es una interfaz orientada a objetos. El modelo de objeto XMS se basa en el modelo de objeto JMS 1.1.

Clases XMS principales

Las clases XMS principales, o tipos de objeto son las siguientes:

ConnectionFactory
Un objeto `ConnectionFactory` encapsula un conjunto de parámetros para una conexión. Una aplicación utiliza un `ConnectionFactory` para crear una conexión. Una aplicación puede proporcionar los parámetros durante el tiempo de ejecución y crear un objeto `ConnectionFactory`. De forma alternativa, los parámetros de conexión se pueden almacenar en un repositorio de objetos administrados. Una aplicación puede recuperar un objeto del repositorio y crear un objeto `ConnectionFactory` a partir del mismo.

Conexión
Un objeto `Connection` encapsula una conexión activa de una aplicación a un servidor de mensajería. Una aplicación utiliza una conexión para crear sesiones.

Destino
Una aplicación envía mensajes o recibe mensajes utilizando un objeto `Destination`. En el dominio de publicación/suscripción, un objeto `Destination` encapsula un tema y, en el dominio punto

a punto, un objeto `Destination` encapsula una cola. Una aplicación puede proporcionar los parámetros para crear un objeto `Destination` durante el tiempo de ejecución. De forma alternativa, puede crear un objeto `Destination` a partir de una definición de objeto que se almacena en el repositorio de objetos administrados.

Session
Un objeto `Session` es un contexto de hebra única para enviar y recibir mensajes. Una aplicación utiliza un objeto `Session` para crear objetos `Message`, `MessageProducer` y `MessageConsumer`.

Mensaje
Un objeto `Message` encapsula el objeto `Message` que envía una aplicación utilizando un objeto `MessageProducer` o que recibe utilizando un objeto `MessageConsumer`.

MessageProducer
Una aplicación utiliza un objeto `MessageProducer` para enviar mensajes a un destino.

MessageConsumer
Una aplicación utiliza un objeto `MessageConsumer` para recibir mensajes enviados a un destino.

Objetos XMS y sus relaciones

Figura 52 en la página 635 muestra los tipos principales de objetos XMS: ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message y Destination. Una aplicación utiliza una fábrica de conexiones para crear una conexión y utiliza una conexión para crear sesiones. La aplicación puede utilizar una sesión para crear mensajes, productores de mensajes y consumidores de mensajes. La aplicación utiliza un productor de mensajes para enviar mensajes a un destino, y utiliza un consumidor de mensajes para recibir mensajes enviados a un destino.

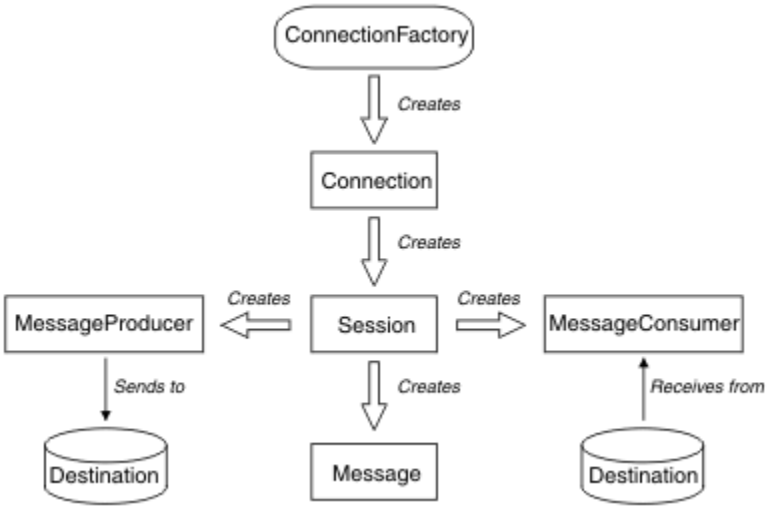


Figura 52. Objetos XMS y sus relaciones

En XMS .NET, las clases XMS se definen como un conjunto de interfaces .NET. Cuando se codifican aplicaciones XMS .NET, solo necesita las interfaces declaradas.

El modelo de objeto XMS se basa en las interfaces independientes de dominio que se describen en la Especificación Java Message Service, versión 1.1. Las clases específicas de dominio como, por ejemplo, `Topic`, `TopicPublisher` y `TopicSubscriber`, no se proporcionan.

Atributos y propiedades de objetos XMS

Un objeto XMS puede tener atributos y propiedades, que son característicos del objeto, que se implementan de formas diferentes:

Atributos
Una característica de objeto que siempre está presente y ocupa almacenamiento, aunque el atributo no tenga un valor. En este sentido, un atributo es similar a un campo en una estructura de datos

de longitud fija. Una característica distintiva de atributos es que cada atributo tiene sus propios métodospara establecer y obtener su valor.

Propiedades
Una propiedad de un objeto está presente y ocupa almacenamiento solo después de que su valor se haya establecido. Una propiedad no se puede suprimir o su almacenamiento se ha recuperado después de que su valor se haya establecido. Puede cambiar su valor. XMS proporciona un conjunto de métodos genéricos para establecer y obtener valores de propiedad.

Objetos administrados

Mediante el uso de objetos administrados, puede administrar los valores de la conexión utilizada por aplicaciones cliente que se van a administrar desde un repositorio central. Una aplicación recupera definiciones de objeto del repositorio central y las utiliza para crear objetos `ConnectionFactory` y `Destination`. Utilizando objetos administrados, puede desacoplar aplicaciones de los recursos que utilizan durante el tiempo de ejecución.

Por ejemplo, las aplicaciones XMS se pueden escribir y probar con objetos administrados que hacen referencia a un conjunto de conexiones y destinos en un entorno de prueba. Cuando se despliegan las aplicaciones, los objetos administrados se pueden cambiar para configurar las aplicaciones para hacer referencia a conexiones y destinos en el entorno de producción.

XMS admite dos tipos de objeto administrado:

- Un objeto `ConnectionFactory`, que es utilizado por aplicaciones para realizar la conexión inicial al servidor.
- Un objeto `Destination`, que es utilizado por aplicaciones para especificar el destino para mensajes que se están enviando, y el origen de mensajes que se están recibiendo. Un destino es un tema o una cola del servidor al que se conecta una aplicación.

La herramienta de administración **`JMSAdmin`** se proporciona con IBM MQ. Se utiliza para crear y gestionar objetos administrados en un repositorio central de objetos administrados.

Los objetos administrados del repositorio pueden ser utilizados por aplicaciones IBM MQ classes for JMS y XMS. Las aplicaciones XMS pueden utilizar los objetos `ConnectionFactory` y `Destination` para conectarse a un IBM MQ gestor de colas. Un administrador puede cambiar las definiciones de objeto incluidas en el repositorio sin que haya repercusiones en el código de aplicación.

El diagrama siguiente muestra cómo una aplicación XMS suele utilizar los objetos administrados. El lado izquierdo del diagrama muestra un repositorio que contiene definiciones de objeto ConnectionFactory y Destination que se administran mediante una consola de administración. El lado derecho del diagrama muestra una aplicación XMS que busca definiciones de objeto del repositorio y, después, utiliza estas definiciones de objeto al conectarse a un servidor de mensajería.

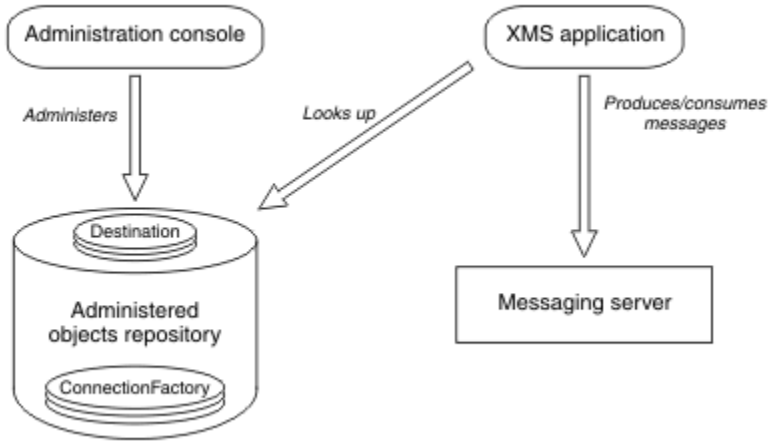


Figura 53. Uso típico de objetos administrados por para de una aplicación XMS

El modelo de mensaje XMS

El modelo de mensaje XMS es el mismo que el modelo de mensaje IBM MQ classes for JMS.

En particular, XMS implementa los mismos campos de cabecera de mensaje y las mismas propiedades de mensaje que implementa IBM MQ classes for JMS:

- Campos de cabecera de JMS. Estos campos tienen nombres que empiezan con el prefijo JMS.
- Propiedades definidas de JMS. Estos campos tienen propiedades cuyos nombres empiezan con el prefijo JMSX.
- Propiedades definidas de IBM. Estos campos tienen propiedades cuyos nombres empiezan con el prefijo JMS_IBM_.

Como resultado, las aplicaciones XMS pueden intercambiar mensajes con aplicaciones IBM MQ classes for JMS. En cada mensaje, algunos de los campos y de las propiedades de cabecera están establecidos por la aplicación y otros están establecidos por XMS o IBM MQ classes for JMS. Algunos de los campos establecidos por XMS o IBM MQ classes for JMS se establecen cuando se envía el mensaje, y otros cuando se recibe. Los campos y las propiedades de cabecera se propagan con un mensaje a través de un servidor de mensajería, cuando proceda. Pasan a estar disponibles para cualquier aplicación que recibe el mensaje.

Conceptos relacionados

IBM MQ classes for JMS

Windows Linux Instalación del IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux.

A partir de IBM MQ 9.2.0, Microsoft.NET Core 3.1 es la versión mínima necesaria para ejecutar IBM MQ classes for XMS .NET Standard.

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, IBM MQ da soporte a aplicaciones .NET 6 utilizando IBM MQ classes for XMS .NET Standard. Si está utilizando una aplicación .NET Core 3.1 , puede ejecutar esta aplicación con una edición pequeña en el archivo `csproj` , estableciendo `targetframeworkversion` en `"net6.0"`, sin que sea necesaria ninguna recompilación.

V 9.3.1 IBM MQ 9.3.1 proporciona una biblioteca de cliente de XMS .NET creada en .NET 6 como infraestructura de destino. A partir de IBM MQ 9.3.1, Microsoft .NET 6.0 es la versión mínima necesaria para ejecutar aplicaciones utilizando bibliotecas de IBM MQ que se compilan utilizando .NET 6 como infraestructura de destino.

V 9.3.1 A partir de IBM MQ 9.3.1, la biblioteca de cliente de XMS .NET creada utilizando .NET Standard está disponible en una carpeta nueva `netstandard2.0` y la biblioteca de cliente de XMS .NET creada utilizando .NET 6 como infraestructura de destino está disponible en *MQ_INSTALLATION_PATH*`/bin` en Windows y en *MQ_INSTALLATION_PATH*`/lib64` en Linux.

Biblioteca `amqmxmsstd.dll`

V 9.3.1 A partir de IBM MQ 9.3.1, la biblioteca `amqmxmsstd.dll` está disponible en las ubicaciones siguientes

Biblioteca creada utilizando .NET Standard 2.0 como infraestructura de destino

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`.
- Linux En Linux: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`.

Deprecated Estas bibliotecas están en desuso y IBM tiene previsto eliminarlas en futuros releases.

Biblioteca creada utilizando .NET 6 como infraestructura de destino

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin`. Las aplicaciones de ejemplo se instalan en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux En Linux: *MQ_INSTALLATION_PATH*`\lib64`. Los ejemplos de .NET se encuentran en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

LTS Para IBM MQ 9.3.0 Long Term Support, la biblioteca de IBM MQ classes for XMS .NET Standard , `amqmxmsstd.dll`, está disponible en las ubicaciones siguientes:

- Windows En Windows: *MQ_INSTALLATION_PATH*`\bin`. Las aplicaciones de ejemplo se instalan en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms`.
- Linux En Linux: *MQ_INSTALLATION_PATH*`/lib64 path`. Los ejemplos de .NET se encuentran en *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/xms`.

Para obtener más información, consulte “Instalación del IBM MQ classes for .NET” en la página 572.

Atención: V 9.3.1 Deprecated A partir de IBM MQ 9.3.1, las bibliotecas de cliente de IBM MQ .NET compiladas utilizando .NET Standard 2.0 como infraestructura de destino están en desuso y las aplicaciones que hacen referencia a estas bibliotecas emiten un aviso CS0618 durante el tiempo de compilación.

Stabilized Aún se proporcionan todas las bibliotecas `IBM.XMS.*`, pero estas bibliotecas se han estabilizado; es decir, no se les añadirán nuevas características. Para cualquiera de las características más recientes, debe migrar a la biblioteca `amqmxmsstd.dll`. Sin embargo, puede seguir utilizando las bibliotecas existentes en los releases IBM MQ 9.1 Long Term Support o Continuous Delivery.

V 9.3.1 Si una aplicación .NET Framework se compila utilizando `amqmdnetstd.dll` o `amqmxmsstd.dll` desde una versión inferior a IBM MQ 9.3.1 y la misma aplicación se ejecuta utilizando las bibliotecas de cliente IBM MQ basadas en .NET 6 , .NETemite el siguiente tipo de excepción `FileLoadException` :

```
Se ha capturado una excepción: System.IO.FileLoadException: No se ha podido cargar el archivo o
ensamblaje
'amqmdnetstd, Version =x.x.x.x, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e' o
una de sus dependencias. La definición de manifiesto del ensamblaje ubicado no coincide con
referencia de ensamblaje. (Excepción de HRESULT: 0x80131040)

Nombre de archivo: ' amqmdnetstd, Version =x.x.x.x, Culture=neutral,
PublicKeyToken=23d6cb914eeaac0e'
```

Para resolver este error, las bibliotecas presentes en *MQ_INSTALLATION_PATH*/bin/netstandard2.0 deben copiarse en el directorio desde donde se ejecuta la aplicación .NET Framework .

A partir de IBM MQ 9.2.0, los IBM MQ classes for XMS .NET Standard están disponibles para su descarga desde el repositorio de NuGet. El paquete NuGet contiene ambas bibliotecas, amqmxmsstd.dll y amqmdnetstd.dll. amqmxmsstd.dll depende de amqmdnetstd.dll y, al empaquetar la aplicación XMS .NET Core, tanto amqmxmsstd.dll como amqmdnetstd.dll deben empaquetarse junto con la aplicación XMS .NET Core. Para obtener más información, consulte "Descarga de IBM MQ classes for XMS .NET desde el repositorio NuGet" en la página 640.

Mandato dspmqver

Puede utilizar el mandato **dspmqver** para visualizar información de versión y compilación para el componente .NET Core .

Comparación entre IBM MQ classes for XMS .NET Framework y IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6)

En la tabla siguiente se listan las características de IBM MQ classes for XMS .NET Framework en comparación con las características de IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6).

Tabla 80. Diferencias entre IBM MQ classes for XMS .NET Framework y IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6)

Característica	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6)
Nombres de clase (API)	Todas las clases siguen siendo las mismas en cada red.	Todas las clases siguen siendo las mismas en cada red.
Sistema operativo	Windows	Windows Contenedores con Docker Linux macOS
Archivo app.config (archivo de configuración para habilitar el rastreo en el cliente redistribuible)	El archivo app.config se utiliza para habilitar el rastreo para el paquete redistribuible.	app.config No recibe soporte. Utilice variables de entorno.
Rastreo	Para rastrear el cliente de XMS .NET , puede utilizar las variables de entorno existentes, como la variable de entorno **XMS_TRACE_ON** utilizada para habilitar el rastreo. Para obtener más información, consulte Configuración del rastreo de XMS .NET utilizando variables de entorno XMS. Para clientes redistribuibles, se puede utilizar el archivo app.config para habilitar el rastreo.	Para rastrear el cliente de XMS .NET , puede utilizar las variables de entorno existentes, como la variable de entorno **XMS_TRACE_ON** utilizada para habilitar el rastreo. Para obtener más información, consulte Configuración del rastreo de XMS .NET utilizando variables de entorno XMS.
Modalidades de transporte	Gestionado, no gestionado y enlaces	Gestionado

Tabla 80. Diferencias entre IBM MQ classes for XMS .NET Framework y IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6) (continuación)

Característica	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET (bibliotecas.NET Standard y .NET 6)
TLS	El almacén de claves Windows se utiliza para almacenar los certificados.	En Windows, se debe utilizar el almacén de claves para almacenar los certificados. Los valores permitidos son *USER o *SYSTEM. Basándose en la entrada, el cliente de IBM MQ .NET busca en el almacén de claves de Windows del usuario actual, o en todo el sistema. En Linux, se recomienda utilizar la clase X509Store para instalar certificados y .NET Core instala certificados en la siguiente ubicación: "`.dotnet/corefx/cryptography/x509stores`".
CCDT	Soportado	Soportado, y los valores de la vía de acceso CCDT son los mismos que para las clases de .NET Framework.
Reconexión automática de cliente	Soportado	Soportado
Transacciones distribuidas	Soportado	No soportado
Instalación de bibliotecas de enlace dinámico (archivos dll) en la memoria caché del conjunto global (GAC)	Los archivos Dll se instalan en la GAC como parte de la instalación de IBM MQ.	Los archivos de Dll no se instalan en la GAC como parte de la instalación de IBM MQ.
Admite los tipos de conexión WMQ, WPM y RTT	Admite los tipos de conexión WMQ, WPM y RTT	Admite solamente WMQ
Objetos administrados JNDI	Admite LDAP y FileSystem	Se admite solamente FileSystem

V 9.3.0 V 9.3.0 A partir de IBM MQ 9.3.0, para ejecutar IBM MQ classes for XMS .NET Framework , debe instalar Microsoft.NET Framework V4.7.2 o posterior.

Tareas relacionadas

“Utilización de las aplicaciones de ejemplo XMS” en la página 647
Las aplicaciones de ejemplo de XMS .NET proporcionan una descripción general de las características comunes de cada API. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Windows Linux Descarga de IBM MQ classes for XMS .NET desde el repositorio NuGet

Los IBM MQ classes for XMS .NET están disponibles para su descarga desde el repositorio de NuGet , para que los desarrolladores de .NET puedan consumirlos fácilmente.

Acerca de esta tarea

NuGet es el gestor de paquetes para las plataformas de desarrollo de Microsoft, incluyendo .NET. Las herramientas de cliente de NuGet proporcionan la capacidad de producir y consumir paquetes. Un paquete NuGet es un único archivo comprimido con la extensión `.nupkg` que contiene código compilado (DLL), otros archivos relacionados con ese código y un manifiesto descriptivo que incluye información como el número de versión del paquete.

Puede descargar el paquete de `IBMXMSDotnetClient` NuGet , que contiene la biblioteca `amqmdnetstd.dll` y la biblioteca `amqmxmsstd.dll` , desde NuGet Gallery, que es el repositorio central de paquetes utilizado por todos los autores y consumidores de paquetes.

Nota: V 9.3.1 A partir de IBM MQ 9.3.1, el paquete NuGet contiene bibliotecas compiladas utilizando .NET Standard 2.0 y .NET 6 como infraestructura de destino. Las bibliotecas para .NET Standard 2.0 están disponibles en la carpeta `netstandard2.0` y las bibliotecas para .NET 6 están disponibles en la carpeta `net6.0` .

Existen tres formas de descarga del paquete `IBMXMSDotnetClient`.

- Utilizando Microsoft Visual Studio. NuGet se distribuye como una extensión de Microsoft Visual Studio. Desde Microsoft Visual Studio 2012, NuGet se instala previamente de forma predeterminada.
- Desde la línea de mandatos, utilizando el gestor de paquetes NuGet, o bien la CLI de .NET.
- Utilizando un navegador web.

Con respecto al paquete redistribuible, habilite el rastreo utilizando la variable de entorno **`XMS_TRACE_ON`**.

Procedimiento

- Para descargar el paquete `IBMXMSDotnetClient` utilizando la interfaz de usuario del gestor de paquetes en Microsoft Visual Studio, complete los pasos siguientes:
  a) Pulse con el botón derecho del ratón en el proyecto .NET y, a continuación, pulse **Gestionar paquetes Nuget**.
  b) Pulse la pestaña **Examinar** y busque `"IBMXMSDotnetClient"`.
  c) Seleccione el paquete y pulse **Instalar**.

  Durante la instalación, el gestor de paquetes proporciona información de progreso en forma de sentencias de consola.
- Para descargar el paquete `IBMXMSDotnetClient` desde la línea de mandatos, elija una de las opciones siguientes:
  - Utilizando el gestor de paquetes NuGet, entre el mandato siguiente:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Durante la instalación, el gestor de paquetes proporciona información de progreso en forma de sentencias de consola. Puede redirigir la salida a un archivo de registro.
  - Utilizando la CLI de .NET, especifique el mandato siguiente:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Utilizando un navegador web, descargue el paquete `IBMXMSDotnetClient` desde https://www.nuget.org/packages/IBMXMSDotnetClient.

Conceptos relacionados

“Instalación del IBM MQ classes for .NET” en la página 572
IBM MQ classes for .NET, incluidos los ejemplos, se instalan con IBM MQ en Windows y Linux

Información de licencia del Cliente de IBM MQ para .NET

Tareas relacionadas

"Descarga de IBM MQ classes for .NET desde el repositorio de NuGet" en la página 578
Los IBM MQ classes for .NET están disponibles para su descarga desde el repositorio de NuGet , para que los desarrolladores de .NET puedan consumirlos fácilmente.

Configuración del entorno de servidor de mensajería

En los temas de esta sección se describe cómo configurar el entorno de servidor de mensajería para permitir a las aplicaciones XMS conectarse a un servidor.

Acerca de esta tarea

Para aplicaciones que se conectan a un gestor de colas IBM MQ, es necesario el cliente de IBM MQ (o el gestor de colas para la modalidad de enlaces).

Actualmente no hay ningún requisito previo para las aplicaciones que utilizan una conexión en tiempo real con un intermediario.

Debe configurar el entorno de servidor de mensajería antes de ejecutar cualquier aplicación XMS, incluyendo las aplicaciones de ejemplo proporcionadas con XMS.

En esta sección se incluyen los temas siguientes:

- "Configuración del gestor de colas y del intermediario para una aplicación que se conecta a un gestor de colas IBM MQ" en la página 644
- "Instalación del IBM MQ classes for XMS .NET" en la página 637
- "Configuración de un intermediario para una aplicación que utiliza una conexión en tiempo real con un intermediario" en la página 645
- "Configuración del bus de integración de servicios para una aplicación que se conecta a WebSphere Application Server" en la página 646

Escuchas de mensajes en XMS .NET

Se utiliza un escucha de mensajes para recibir mensajes de forma asíncrona. A diferencia de la llamada `MessageConsumer.receive()` , el escucha de mensajes no bloquea la hebra de llamada, sino que entrega mensajes a un método de devolución de llamada especificado por la aplicación, normalmente el método `onMessage` .

La entrega de mensajes se inicia una vez que se llama al método `Connection.Start()` . La entrega de mensajes se puede detener y reanudar en cualquier momento utilizando los métodos `Connection.Stop()` y `Connection.Start()` respectivamente.

Una vez que se llama al método `Connection.Start()` después de establecer un escucha de mensajes en al menos un consumidor en una sesión, dicha sesión se convierte en una sesión asíncrona. Una vez que una sesión se vuelve asíncrona, no es posible llamar a ningún método síncrono XMS .NET ., Por ejemplo, `MessageProducer.Send()`. Al hacerlo, se produce una excepción con el código de razón IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

Llamadas síncronas en una sesión asíncrona

`Session.Close` es la única llamada síncrona permitida en una sesión asíncrona. Las aplicaciones también pueden realizar llamadas síncronas (excepto `Session.Close`) utilizando el método de devolución de llamada de escucha de mensajes, es decir, el método `onMessage` .

Aparte de estas dos opciones, debe detener la conexión utilizando el método `Connection.Stop()` para que una aplicación realice cualquier llamada síncrona. Después de realizar las llamadas, debe reanudar la conexión de nuevo utilizando el método `Connection.Start()` . que reinicia la entrega de mensajes.

¿Cuántos consumidores de mensajes asíncronos puede tener una sesión?

Una sesión puede tener varios consumidores de mensajes asíncronos. Pero en cualquier momento un mensaje se entrega a un solo consumidor. Lo que esto significa prácticamente es que, cuando llega un segundo mensaje mientras XMS .NET ha llamado al método `onMessage()` de un consumidor para entregar el primer mensaje, el segundo mensaje no se entregará a un consumidor en la sesión hasta que se devuelva el método `onMessage()` .

El segundo mensaje se entrega a un consumidor en la sesión sólo después de que se devuelva el método `onMessage()` . Esto se debe a que una sesión gestiona la entrega de mensajes a los consumidores utilizando sólo una hebra. Esto significa que solo se puede entregar un mensaje a la vez, y el consumidor podría ser cualquiera.

Si una aplicación requiere la entrega simultánea de mensajes, es decir, todos los consumidores deben recibir mensajes al mismo tiempo, la aplicación debe crear varias sesiones y cada una debe tener un consumidor de mensajes asíncrono.

Los ejemplos siguientes muestran esta característica de forma más clara.

En el primer ejemplo, hay varios consumidores de mensajes asíncronos en una sesión. Una sesión S tiene tres consumidores de mensajes asíncronos: AMC1, AMC2 y AMC3 que reciben mensajes de tres destinos diferentes Q1, Q2 y Q3.

Como sólo hay una sesión S, sólo hay una hebra de entrega de mensajes para entregar mensajes a los consumidores `AMC1`, `AMC2`y `AMC3`. Cuando la sesión está entregando mensajes a `AMC1`, los otros dos consumidores `AMC2` y `AMC3` esperan, incluso si hay mensajes en `Q2` y `Q3` listos para su entrega.

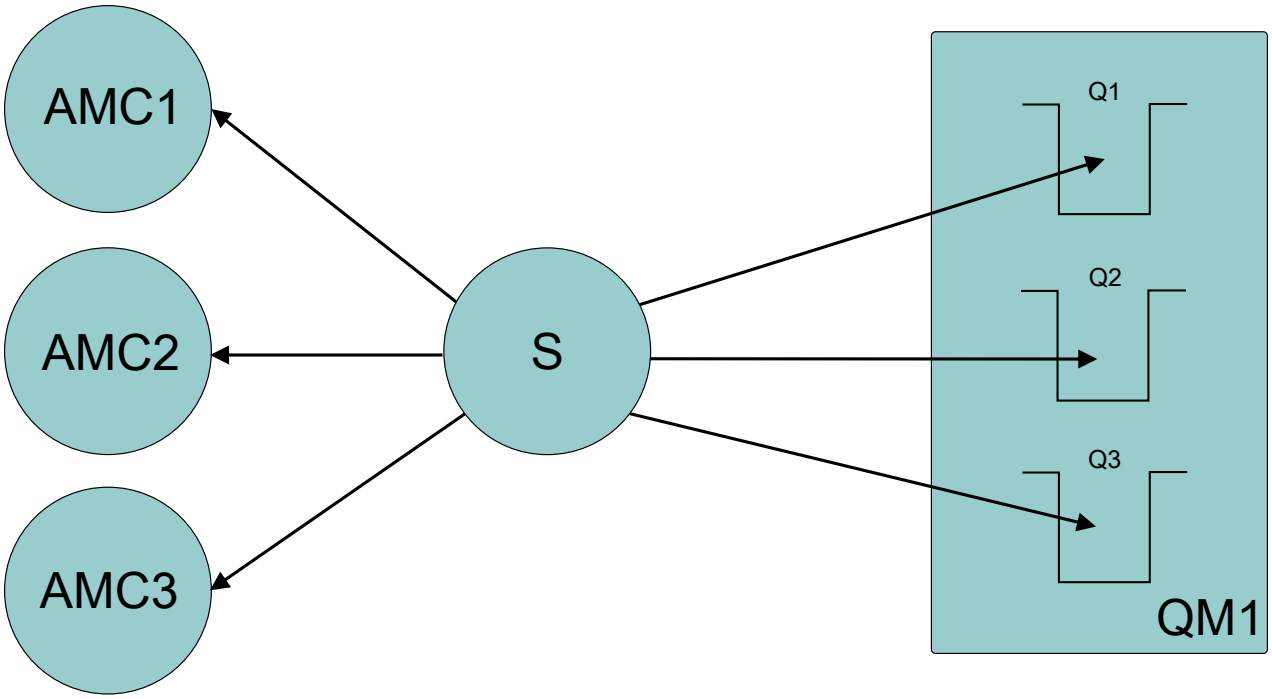


Figura 54. Una sesión con tres consumidores de mensajes asíncronos

En el segundo caso, hay varias sesiones S1, S2y S3, cada una con un consumidor de mensajes asíncrono `AMC1`, `AMC2`y `AMC3` respectivamente. Como hay un consumidor para cada sesión, los mensajes se entregan a los consumidores de forma simultánea.

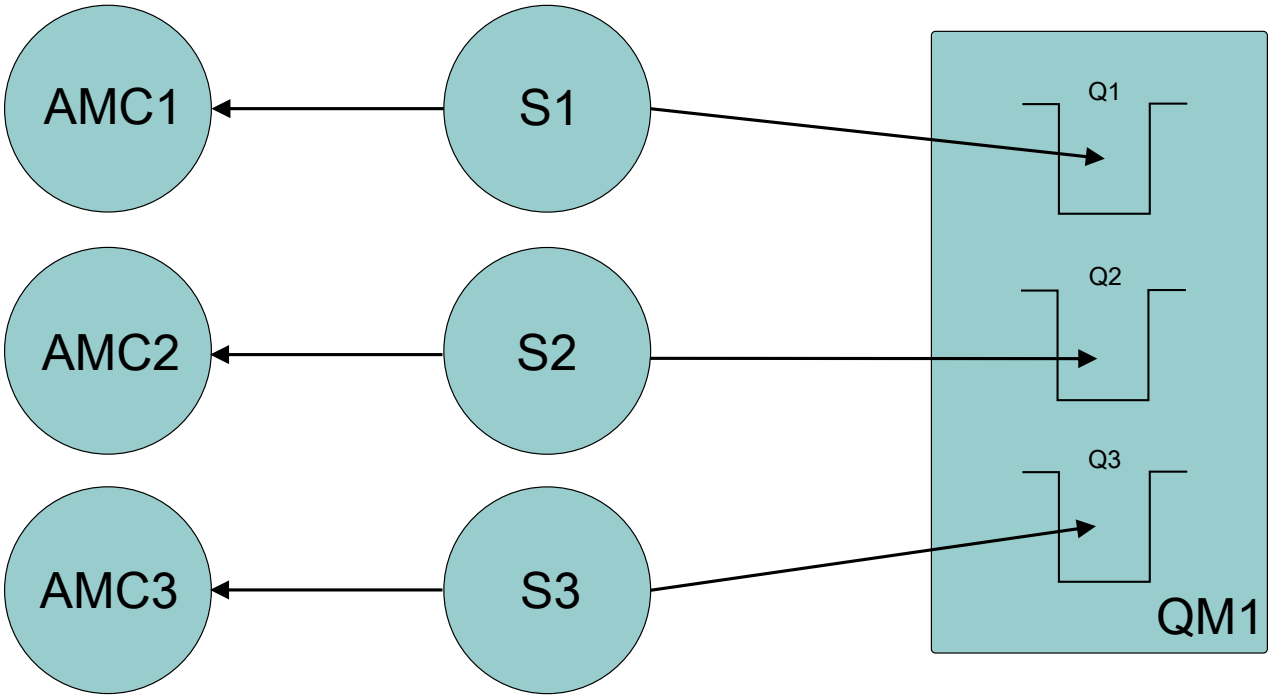


Figura 55. Varias sesiones, cada una con un consumidor de mensajes asíncrono

Esto muestra que si necesita la entrega simultánea de mensajes, necesita varias sesiones.

Configuración del gestor de colas y del intermediario para una aplicación que se conecta a un gestor de colas IBM MQ

En esta sección se da por supuesto que está utilizando IBM WebSphere MQ 7.0.1, o posterior. Antes de poder ejecutar una aplicación que se conecta a un gestor de colas IBM MQ, debe configurar el gestor de colas. Para una aplicación de publicación/suscripción, es necesaria alguna configuración adicional, si está utilizando la interfaz de publicación/suscripción en cola.

Antes de empezar

XMS funciona con IBM Integration Bus o WebSphere Message Broker 6.1 o posterior

Antes de iniciar esta tarea, realice los pasos siguientes:

- Asegúrese de que la aplicación tiene acceso a un gestor de colas que se está ejecutando.
- Si la aplicación es una aplicación de publicación/suscripción y utiliza la interfaz de publicación/suscripción en cola, asegúrese de que el atributo **PSMODE** está establecido en ENABLED en el gestor de colas.
- Asegúrese de que la aplicación utiliza una fábrica de conexiones cuyas propiedades se han establecido correctamente para conectarse al gestor de colas. Si la aplicación es una aplicación de publicación/suscripción, asegúrese de que las propiedades de fábrica de conexiones apropiadas se establecen para utilizar el intermediario. Para obtener más información sobre las propiedades de una fábrica de conexiones, consulte Propiedades de ConnectionFactory.

Acerca de esta tarea

Configure el gestor de colas y el intermediario para ejecutar aplicaciones XMS de la misma forma en la que se configura el gestor de colas y la interfaz de publicación/suscripción en cola para ejecutar aplicaciones IBM MQ JMS. Los pasos siguientes resumen lo qué debe hacer.

Procedimiento

1. En el gestor de colas, cree las colas que necesita la aplicación.

   Para obtener una descripción general sobre cómo crear colas, consulte Definición de colas.

   Si la aplicación es una aplicación de publicación/suscripción y utiliza la interfaz de publicación/suscripción en cola que necesita acceder a colas del sistema IBM MQ classes for JMS, espere hasta el paso 4a antes de crear las colas.
2. Otorgue al ID de usuario asociado a la aplicación la autoridad para conectarse al gestor de colas, y la autoridad apropiada para acceder a las colas.

   Para obtener una descripción general sobre la autorización, consulte Protección. Si la aplicación se conecta al gestor de colas en la modalidad de cliente, consulte también Clientes y servidores.
3. Si la aplicación se conecta al gestor de colas en la modalidad de cliente, asegúrese de que un canal de conexión de servidor está definido en el gestor de colas y de que se ha iniciado un escucha.

   No tendrá que realizar este paso para cada aplicación que se conecta al gestor de colas. Una definición de canal de conexión de servidor y un escucha pueden dar soporte a todas las aplicaciones que se conectan en la modalidad de cliente.
4. Si la aplicación es una aplicación de publicación/suscripción y utiliza la interfaz de publicación/suscripción en cola, realice los pasos siguientes.
   a) En el gestor de colas, cree las colas del sistema IBM MQ classes for JMS ejecutando el script de mandatosMQSC que se proporcionan con IBM MQ. Asegúrese de que el ID de usuario asociado a IBM Integration Bus o WebSphere Message Broker tiene la autoridad para acceder a las colas.

      Si desea más información sobre dónde encontrar el script y cómo ejecutarlo, consulte Utilización de IBM MQ classes for Java.

      Realice este paso solo una vez para el gestor de colas. El mismo conjunto de colas del sistema IBM MQ classes for JMS puede dar soporte a todas las aplicaciones XMS y IBM MQ classes for JMS que se conectan al gestor de colas.
   b) Otorgue al ID de usuario asociado a la aplicación la autoridad para acceder a las colas del sistema IBM MQ classes for JMS.

      Si desea más información sobre qué autoridades necesita el ID de usuario, consulte Utilización de IBM MQ classes for JMS.
   c) Para un intermediario de IBM Integration Bus o WebSphere Message Broker, cree y despliegue un flujo de mensajes para prestar servicio a la cola donde las aplicaciones envían mensajes que publican.

      El flujo de mensajes básico engloba un nodo de proceso de mensajes MQInput para leer los mensajes publicados y un nodo de proceso de mensajes Publication para publicar los mensajes.

      Si desea más información sobre cómo crear y desplegar un flujo de mensajes, consulte la documentación del producto IBM Integration Bus o WebSphere Message Broker disponible desde IBM Integration Bus página web de la biblioteca de documentación del producto.

      No es necesario que realice este paso si ya se ha desplegado un flujo de mensajes apropiado en el intermediario.

Resultados

Ahora puede iniciar la aplicación.

Configuración de un intermediario para una aplicación que utiliza una conexión en tiempo real con un intermediario

Antes de poder ejecutar una aplicación que utiliza una conexión en tiempo real con un intermediario, debe configurar ese intermediario.

Antes de empezar

Antes de iniciar esta tarea, realice los pasos siguientes:

- Asegúrese de que la aplicación tiene acceso a un intermediario que se está ejecutando.
- Asegúrese de que la aplicación utiliza una fábrica de conexiones cuyas propiedades se han establecido de forma adecuada para una conexión en tiempo real con un intermediario. Para obtener más información sobre las propiedades de una fábrica de conexiones, consulte Propiedades de ConnectionFactory.

Acerca de esta tarea

Configure un intermediario para ejecutar aplicaciones XMS de la misma forma que se configura un intermediario para ejecutar aplicaciones IBM MQ classes for JMS. Los pasos siguientes resumen lo qué debe hacer:

Procedimiento

1. Crear y desplegar un flujo de mensajes para leer mensajes desde el puerto TCP/IP en el cual escucha un intermediario y publica los mensajes.

   Puede hacerlo de cualquiera de estas formas:

   - Crear un flujo de mensajes que contenga un nodo de proceso de mensajes **Real-timeOptimizedFlow**.
   - Crear un flujo de mensajes que contenga un nodo de proceso de mensajes **Real-timeInput** y un nodo de proceso de mensajes Publication.

   Debe configurar el nodo **Real-timeOptimizedFlow** o **Real-timeInput** para escuchar en el puerto utilizado para conexiones en tiempo real. En XMS, el número de puerto predeterminado para conexiones en tiempo real es 1506.

   No es necesario que realice este paso si ya se ha desplegado un flujo de mensajes apropiado en el intermediario.
2. Si requiere que los mensajes se entreguen en la aplicación mediante IBM MQ classes for JMS, configure el intermediario para habilitar la multidifusión. Configure los temas que deben tener la multidifusión habilitada, especificando una calidad de servicio fiable para estos temas que requieren una multidifusión fiable.
3. Si la aplicación proporciona un ID de usuario y una contraseña cuando se conecta a un intermediario, y desea que el intermediario autentique su aplicación utilizando esta información, configure el servidor de nombres de usuario y el intermediario para una autenticación de contraseña simple similar al telnet.

Resultados

Ahora puede iniciar la aplicación.

Configuración del bus de integración de servicios para una aplicación que se conecta a WebSphere Application Server

Antes de poder ejecutar una aplicación que se conecta a un bus de integración de servicios WebSphere Application Server service integration technologies, debe configurar la integración de servicio de la misma forma que configura el bus de integración de servicios para ejecutar aplicaciones JMS que utilizan el proveedor de mensajería predeterminado.

Antes de empezar

Antes de iniciar esta tarea, debe realizar los pasos siguientes:

- Asegúrese de que se crea un bus de mensajería y que el servidor se ha añadido al bus como miembro del bus.

- Asegúrese de que la aplicación tiene acceso a un bus de integración de servicios que contiene, al menos, un motor de mensajería que se está ejecutando.
- Si es necesario el funcionamiento de HTTP, se debe definir un canal de transporte de entrada de motor de mensajería de HTTP. De forma predeterminada, los canales para SSL y TCP se definen durante la instalación del servidor.
- Asegúrese de que la aplicación utiliza una fábrica de conexiones cuyas propiedades se han establecido de forma apropiada para conectarse al bus de integración de servicios utilizando un servidor de programa de arranque. La información mínima necesaria es:
  - El punto final de proveedor, que describe la ubicación y el protocolo para utilizar al negociar una conexión al servidor de mensajería (es decir, a través del servidor de programa de arranque). En su forma más sencilla, para un servidor instalado con valores predeterminados, el punto final de proveedor se puede establecer en el nombre de host del servidor.
  - El nombre del bus a través del cual se envían los mensajes.

  Para obtener más información sobre las propiedades de una fábrica de conexiones, consulte Propiedades de ConnectionFactory.

Acerca de esta tarea

Cualquier cola o espacio de tema que necesite debe estar definido. De forma predeterminada, un espacio de tema llamado Default.Topic.Space se define durante la instalación del servidor pero, si necesita más espacios de tema, debe crear estos espacios de tema usted mismo. No es necesario predefinir temas individuales en un espacio de tema, porque el servidor crea una instancia de estos temas individuales de forma dinámica, según sea necesario.

Los pasos siguientes resumen lo qué debe hacer.

Procedimiento

1. Cree las colas que necesita la aplicación para la mensajería punto a punto.
2. Cree cualquier espacio de tema adicional que necesita la aplicación para la mensajería de publicación/suscripción.

Resultados

Ahora puede iniciar la aplicación.

Utilización de las aplicaciones de ejemplo XMS

Las aplicaciones de ejemplo de XMS .NET proporcionan una descripción general de las características comunes de cada API. Puede utilizarlas para verificar la instalación y la configuración del servidor de mensajería y para ayudarle a crear sus propias aplicaciones.

Acerca de esta tarea

Si necesita ayuda para crear sus propias aplicaciones, puede utilizar las aplicaciones de ejemplo como punto de partida. Se proporcionan ambas versiones, la de origen y la compilada, para cada aplicación. Revise el código fuente de ejemplo e identifique los pasos clave para crear cada objeto necesario para la aplicación (ConnectionFactory, Connection, Session, Destination, y un Producer, or un Consumer, o ambos), y para establecer cualquier propiedad específica necesaria para especificar cómo desea que funcione la aplicación. Para obtener más información, consulte “Cómo escribir aplicaciones de XMS” en la página 650. Los ejemplos están sujetos a cambios en futuros releases de XMS.

La tabla siguiente muestra los conjuntos de aplicaciones de ejemplo (uno para cada API) que se proporcionan con XMS.

Tabla 81. Aplicaciones de ejemplo para XMS.NET	
Nombre de ejemplo	**Descripción**
SampleConsumerCS	Una aplicación de consumidor de mensajes que toma mensajes de una cola o que se suscribe a un tema.
SampleProducerCS	Una aplicación de productor de mensajes que genera mensajes en una cola o sobre un tema.
SampleConfigCS	Una aplicación de configuración que puede utilizar para crear un repositorio de objetos administradores que se basa en archivo. La aplicación contiene una fábrica de conexiones y un destino para sus valores de conexión en particular. Este repositorio de objetos administrados se puede utilizar después con cada una de las aplicaciones de productor y consumidor de ejemplo.

Los ejemplos que dan soporte a las mismas funciones den las distintas API tienen diferencias sintácticas.

- Las aplicaciones de productor y consumidor de mensajes de ejemplo soportan ambas las funciones siguientes:
  - Conexiones a IBM MQ, IBM Integration Bus (utilizando una conexión en tiempo real a un intermediario) y un WebSphere Application Server service integration bus
  - Búsquedas de repositorio de objetos administrados utilizando la interfaz de contexto inicial
  - Conexiones a colas (IBM MQ y WebSphere Application Server service integration bus) y temas (IBM MQ, conexión en tiempo real a un intermediario, y WebSphere Application Server service integration bus)
  - Mensajes base, de byte, de correlación, de objeto, de corriente de datos y de texto
- La aplicación de consumidor de mensajes de ejemplo admite las modalidades de recepción síncrona y asíncrona y sentencias de SQL Selector.
- La aplicación de productor de mensajes de ejemplo admite las modalidades de entrega persistente y no persistente.

Los ejemplos pueden funcionar de una de estas dos modalidades:

Modalidad simple
Puede ejecutar los ejemplos con la mínima entrada de usuario.

Modalidad avanzada
Puede personalizar de forma más precisa la forma en la que funcionan los ejemplos.

Todos los ejemplos son compatibles y, por lo tanto, pueden funcionar entre lenguajes.

Windows A partir de IBM MQ 9.1.1, IBM MQ admite .NET Core para aplicaciones XMS .NET en entornos Windows. IBM MQ classes for .NET Standard, incluidos los ejemplos, se instalan de forma predeterminada como parte de la instalación estándar de IBM MQ.

Linux A partir de IBM MQ 9.1.2, IBM MQ también admite .NET Core para aplicaciones en entornos Linux.

Las aplicaciones de ejemplo de XMS .NET se instalan en `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Para obtener más información, consulte “Instalación del IBM MQ classes for XMS .NET” en la página 637.

Ejecución de aplicaciones de ejemplo de .NET

Puede ejecutar las aplicaciones de ejemplo de .NET de forma interactiva en la modalidad simple o avanzada, o no interactiva utilizando archivos de respuestas generados automáticamente o personalizados.

Antes de empezar

Antes de ejecutar cualquiera de las aplicaciones de ejemplo proporcionadas, primero debe configurar el entorno del servidor de mensajería para que las aplicaciones se puedan conectar a un servidor. Consulte “Configuración del entorno de servidor de mensajería” en la página 642.

Procedimiento

Para ejecutar una aplicación de ejemplo .NET, complete los pasos siguientes:

Consejo: Cuando se ejecuta una aplicación de ejemplo, escriba ? en cualquier momento para obtener ayuda sobre qué hacer a continuación.

1. Seleccione la modalidad en la cual desea ejecutar la aplicación de ejemplo.

   Escriba `Advanced` o `Simple`.
2. Responda las preguntas.

   Para seleccionar el valor predeterminado, que se muestra en los corchetes al final de la pregunta, pulse Intro. Para seleccionar un valor diferente, escriba el valor apropiado y pulse Intro.

   A continuación, se muestra una pregunta de ejemplo:

   ```
   Enter connection type [wpm]:
   ```

   En este caso, el valor predeterminado es `wpm` (conexión a un WebSphere Application Server service integration bus).

Resultados

Cuando se ejecutan las aplicaciones de ejemplo, los archivos de respuestas se generan automáticamente en el directorio de trabajo actual. Los nombres de archivo de respuestas tienen el formato *`connection_type-sample_type`*`.rsp`; por ejemplo, `wpm-producer.rsp`. Si es necesario, puede utilizar el archivo de respuestas generado para volver a ejecutar la aplicación de ejemplo con las mismas opciones, de modo que no tenga que volver a entrar las opciones.

Tareas relacionadas

Creación de aplicaciones de ejemplo de .NET
Cuando se crea una aplicación de ejemplo de .NET, se crea una versión ejecutable del ejemplo seleccionado.

Creación de sus propias aplicaciones
Cree sus propias aplicaciones como crea las aplicaciones de ejemplo.

Creación de aplicaciones de ejemplo de .NET

Cuando se crea una aplicación de ejemplo de .NET, se crea una versión ejecutable del ejemplo seleccionado.

Antes de empezar

Instale el compilador apropiado. Esta tarea presupone que ha instalado Microsoft Visual Studio 2012 y que está familiarizado con su uso.

Procedimiento

Para crear una aplicación de ejemplo .NET, complete los pasos siguientes:

1. Pulse el archivo de solución `Samples.sln` proporcionado con los ejemplos de .NET.
2. Pulse con el botón derecho del ratón en la solución `Samples` en la ventana del Explorador de soluciones y seleccione **Crear solución**.

Resultados

Se crea un programa ejecutable en la subcarpeta apropiada del ejemplo, `bin/Debug` o `bin/Release`, según la configuración que haya elegido. Este programa tiene el mismo nombre que la carpeta, con un sufijo de CS. Por ejemplo, si está creando la versión C# de la aplicación de ejemplo del productos de mensajes, se crea `SampleProducerCS.exe` en la carpeta `SampleProducer`.

Tareas relacionadas

Ejecución de aplicaciones de ejemplo de .NET
Puede ejecutar las aplicaciones de ejemplo de .NET de forma interactiva en la modalidad simple o avanzada, o no interactiva utilizando archivos de respuestas generados automáticamente o personalizados.

Creación de sus propias aplicaciones
Cree sus propias aplicaciones como crea las aplicaciones de ejemplo.

"Creación de sus propias aplicaciones" en la página 650
Cree sus propias aplicaciones como crea las aplicaciones de ejemplo.

Creación de sus propias aplicaciones

Cree sus propias aplicaciones como crea las aplicaciones de ejemplo.

Antes de empezar

Instale el compilador apropiado. Esta tarea presupone que ha instalado Microsoft Visual Studio 2012 y que está familiarizado con su uso.

Procedimiento

- Cree la aplicación .NET, tal como se describe en "Creación de aplicaciones de ejemplo de .NET" en la página 649.

  Si desea ayuda adicional para crear sus propias aplicaciones, utilice los archivos makefiles proporcionados para cada aplicación de ejemplo.

  Consejo: Para ayudarle con el diagnóstico de problemas en el supuesto de una anomalía, es posible que encuentre útil compilar las aplicaciones con los símbolos incluidos.

Tareas relacionadas

Ejecución de aplicaciones de ejemplo de .NET
Puede ejecutar las aplicaciones de ejemplo de .NET de forma interactiva en la modalidad simple o avanzada, o no interactiva utilizando archivos de respuestas generados automáticamente o personalizados.

Creación de aplicaciones de ejemplo de .NET
Cuando se crea una aplicación de ejemplo de .NET, se crea una versión ejecutable del ejemplo seleccionado.

Cómo escribir aplicaciones de XMS

Los temas de esta sección proporcionan información que le ayudará a escribir aplicaciones XMS en general.

Acerca de esta tarea

En esta sección se incluyen conceptos generales para escribir aplicaciones de XMS. Consulte también "Escritura de aplicaciones de XMS .NET" en la página 669 si desea información específica para crear aplicaciones de XMS .NET.

Desde IBM MQ 9.2.0, el número de bibliotecas de enlace dinámico XMS.NET se ha reducido significativamente, hasta un total de cinco. Las cinco bibliotecas de enlace dinámico son:

- `IBM.XMS.dll` - incluye todos los mensajes de idioma nacional

- `IBM.XMS.Comms.RMM.dll`
- Tres bibliotecas de enlace dinámico de políticas:
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

V 9.3.0 Removed V 9.3.0 XMS .NET La mensajería de multidifusión (mediante RMM) está en desuso desde IBM MQ 9.2 y se eliminó en IBM MQ 9.3.

En esta sección se incluyen los temas siguientes:

- "El modelo de agrupación en hebras" en la página 652
- "Objetos ConnectionFactories y Connection" en la página 652
- "Sesiones" en la página 654
- "Destinos" en la página 657
- "Productores de mensajes" en la página 660
- "Consumidores de mensajes" en la página 660
- "Examinadores de colas" en la página 664
- "Solicitantes" en la página 664
- "Supresión de objeto" en la página 665
- "Tipos primitivos de XMS" en la página 666
- "Conversión implícita de un valor de propiedad de un tipo de datos a otro" en la página 666
- "Iteradores" en la página 668
- "Identificadores de juego de caracteres codificado" en la página 669
- "Códigos de error y excepción de XMS" en la página 669
- "Creación de sus propias aplicaciones" en la página 650

Windows Utilización de la plantilla de proyecto de IBM MQ XMS .NET

El cliente de IBM MQ XMS .NET le ofrece la posibilidad de utilizar una plantilla de proyecto para ayudarle a desarrollar sus aplicaciones XMS .NET Core .

Antes de empezar

Debe tener Microsoft Visual Studio 2017, o posterior, y .NET Core 2.1 en el sistema.

Debe copiar la plantilla XMS .NET de la

`&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip`

al directorio

`&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates`

directorio, donde:

- *& MQ_INSTALL_ROOT* es el directorio raíz de la instalación
- *& USER_HOME_DIRECTORY* es el directorio de inicio.

Debe detener y reiniciar Microsoft Visual Studio para seleccionar la plantilla.

Acerca de esta tarea

La plantilla de proyecto XMS .NET incluye algún código común que puede utilizar para ayudar a desarrollar las aplicaciones. Con el código incorporado, puede conectarse al gestor de colas IBM MQ y realizar una operación de colocación u obtención simplemente modificando las propiedades del código incorporado.

Procedimiento

1. Abra Microsoft Visual Studio.
2. Pulse en **Archivo**, seguido por **Nuevo** y, a continuación, **Proyecto**.
3. En la ventana *Crear un proyecto nuevo*, seleccione `IBM XMS .NET Client App (.NET Core)` y pulse **Siguiente**.
4. En la ventana *Configurar el proyecto nuevo*, cambie el *Nombre de proyecto* del proyecto si lo desea y pulse **Crear** para crear el proyecto XMS .NET.

   `XMSDotnetApp.cs` es el archivo que se crea junto con el archivo de proyecto. Este archivo contiene el código que se conecta al gestor de colas y realiza una operación de enviar y recibir.

   Las propiedades de conexión se establecen en valores predeterminados:

   - WMQ_CONNECTION_NAME_LIST se establece en *localhost(1414)*
   - XMSC.WMQ_CHANNEL se establece en *DOTNET.SVRCONN*

   La cola se establece en *Q1*, y puede modificar estas propiedades en consecuencia.
5. Compile y ejecute la aplicación.

Conceptos relacionados

Componentes y características de IBM MQ

Tiempo de ejecución de la aplicación .NET - Solamente en Windows

El modelo de agrupación en hebras

Reglas generales controlan cómo una aplicación de varias hebras puede utilizar objetos XMS.

- Solo los objetos de los tipos siguientes se pueden utilizar de forma simultánea en hebras distintas:
  - ConnectionFactory
  - Conexión
  - ConnectionMetaData
  - Destino
- Un objeto Session se puede utilizar en solo una sola hebra a la vez.

Las excepciones de estas reglas se indican mediante entradas con la etiqueta "Contexto de hebra" en las definiciones de interfaz de los métodos en Referencia de IBM Message Service Client para .NET.

Objetos ConnectionFactories y Connection

Un objeto ConnectionFactory proporciona una plantilla que utiliza una aplicación para crear un objeto Connection. La aplicación utiliza el objeto Connection para crear un objeto Session.

Para .NET, la aplicación XMS primero utiliza un objeto XMSFactoryFactory para obtener una referencia a un objeto ConnectionFactory que sea apropiado para el tipo de protocolo necesario. Este objeto ConnectionFactory puede generar conexiones solo para ese tipo de protocolo.

Una aplicación XMS puede crear varias conexiones, y una aplicación de varias hebras puede utilizar un solo objeto Connection de forma simultánea en varias hebras. Un objeto Connection encapsula una conexión de comunicaciones entre una aplicación y un servidor de mensajería.

Una conexión sirva para varias finalidades:

- Cuando una aplicación crea una conexión, la aplicación se puede autenticar.

- Una aplicación puede asociar un identificador de cliente exclusivo a una conexión. El identificador de cliente se utiliza para dar soporte a suscripciones duraderas en el dominio de publicación/suscripción. El identificador de cliente se puede establecer de dos maneras:

  La forma preferida de asignar un identificador de cliente de conexiones es configurar en un objeto ConnectionFactory específico de cliente utilizando propiedades y asignarlo de forma transparente a la conexión que crea.

  Una forma alternativa de asignar un identificador de cliente es utilizar un valor específico del proveedor que se establece en el objeto Connection. Este valor no sustituye el identificador que se ha configurado de forma administrativa. Se proporciona para el caso donde no existe ningún identificador especificado de forma administrativa. Si un identificador especificado de forma administrativa no existe, un intento de sustituirlo con un valor específico de proveedor provoca que se lance una excepción. Si una aplicación define de forma explícita un identificador, debe hacerlo inmediatamente después de crear la conexión y antes de que se realice cualquier otra acción en la conexión; de lo contrario, se lanza una excepción.

Normalmente, una aplicación XMS crea una conexión, una o más sesiones, y un número de productores de mensajes y consumidores de mensajes.

La creación de una conexión es relativamente cara en términos de recursos del sistema, porque implica el establecimiento de una conexión de comunicaciones y, también, podría implicar la autenticación de la aplicación.

Modalidad iniciada y detenida de conexión

Una conexión puede operar tanto en modalidad iniciada como detenida.

Cuando una aplicación crea una conexión, la conexión está en la modalidad detenida. Cuando la conexión está en la modalidad detenida, la aplicación puede inicializar sesiones, y puede enviar mensajes pero no puede recibirlos, ya sea de forma síncrona o asíncrona.

Una aplicación puede iniciar una conexión llamando al método `Start Connection` . Cuando la conexión está en la modalidad iniciada, la aplicación puede enviar y recibir mensajes. A continuación, la aplicación puede detener y reiniciar la conexión llamando a los métodos Detener conexión y `Start Connection` .

Cierre de conexión

Una aplicación cierra una conexión llamando al método Cerrar conexión. Cuando una aplicación cierra una conexión, XMS realiza las acciones siguientes:

- Cierra todas las sesiones asociadas a la conexión y suprime determinados objetos asociados a estas sesiones. Si desea más información sobre qué objetos se suprimen, consulte “Supresión de objeto” en la página 665. Al mismo tiempo, XMS retrotrae las transacciones actualmente en curso dentro de las sesiones.
- Finaliza la conexión de comunicaciones con el servidor de mensajería.
- Libera la memoria y otros recursos internos utilizados por la conexión.

XMS no crea ningún acuse de recibo de la recepción de ninguno de los mensajes que no ha podido reconocer durante una sesión, antes de cerrar la conexión. Si desea más información sobre cómo crear un acusar recibo de la recepción de mensajes, consulte “Acuse de recibo de mensaje” en la página 655.

Manejo de excepciones

Todas las excepciones de XMS .NET se han derivado de System.Exception. Para obtener más información, consulte “Manejo de errores en .NET” en la página 673.

Conexión a un bus de integración de servicios

Una aplicación XMS puede conectarse a un bus de integración de servicios WebSphere Application Server utilizando una conexión TCP/IP directa o utilizando HTTP sobre TCP/IP.

El protocolo HTTP se puede utilizar en situaciones donde no es posible una conexión TCP/IP directa. Una situación común es cuando se comunican a través de un cortafuegos como, por ejemplo, cuando dos empresas intercambian mensajes. A menudo, el uso de HTTP para comunicarse a través de una cortafuegos se denomina como *ejecución en túnel HTTP*. Sin embargo, la ejecución en túnel HTTP es inherentemente más lenta que el uso de la conexión TCP/IP directa porque las cabeceras HTTP añaden una cantidad de datos significativa que se transfieren y porque el protocolo HTTP requiere más flujos de comunicación que TCP/IP.

Para crear una conexión TCP/IP, una aplicación puede utilizar una fábrica de conexiones cuya propiedad XMSC_WPM_TARGET_TRANSPORT_CHAIN está establecida en XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Este es el valor predeterminado de la propiedad. Si la conexión se crea correctamente, la propiedad XMSC_WPM_CONNECTION_PROTOCOL de la conexión se establece en XMSC_WPM_CP_TCP.

Para crear una conexión que utiliza HTTP, una aplicación debe utilizar una fábrica de conexiones cuya propiedad XMSC_WPM_TARGET_TRANSPORT_CHAIN se establece en el nombre de una cadena de transporte de salida, que se ha configurado para utilizar un canal de transporte HTTP. Si la conexión se crea correctamente, la propiedad XMSC_WPM_CONNECTION_PROTOCOL de la conexión se establece en XMSC_WPM_CP_HTTP. Si desea más información sobre cómo configurar cadenas de transporte, consulte Configuración de cadenas de transporte en la documentación del producto WebSphere Application Server.

Una aplicación tiene una opción similar de protocolos de comunicaciones al conectarse a un servidor de programa de arranque. La propiedad XMSC_WPM_PROVIDER_ENDPOINTS de una fábrica de conexiones es una secuencia de una o más direcciones de punto final de servidores de programa de arranque. El componente de cadena de transporte del programa de arranque de cada dirección de punto final puede ser XMSC_WPM_BOOTSTRAP_TCP, para una conexión de TCP/IP a un servidor de programa de arranque o XMSC_WPM_BOOTSTRAP_HTTP, para una conexión que utiliza HTTP.

Sesiones

Una sesión es un contexto de hebra única para enviar y recibir mensajes.

Una aplicación puede utilizar una sesión para crear mensajes, productores de mensajes, consumidores de mensajes, navegadores de colas y destinos temporales. Una aplicación también puede utilizar una sesión para ejecutar transacciones locales.

Una aplicación puede crear varias sesiones, donde cada sesión produce y consume mensajes independientemente de las otras sesiones. Si dos consumidores de mensajes en sesiones separadas (o incluso en la misma sesión) se suscriben al mismo tema, cada uno recibe una copia de cualquier mensaje publicado sobre dicho tema.

A diferencia de un objeto Connection, un objeto Session no se puede utilizar de forma simultánea en hebras diferentes. Solo el método Cerrar sesión de un objeto Session se puede llamar desde una hebra distinta a la que está utilizando el objeto Session en ese momento. El método Cerrar sesión finaliza una sesión y libera cualquier recurso del sistema asignado a la sesión.

Si una aplicación debe procesar mensajes de forma simultánea en más de una hebra, la aplicación debe crear una sesión en cada hebra y, después, utilizar esa sesión para cualquier operación enviar o recibir en dicha hebra.

Sesiones con transacción

Las aplicaciones XMS pueden ejecutar transacciones locales. Una *transacción local* es una transacción que implica cambios solo en los recursos del gestor de colas o el bus de integración de servicios al cual está conectada la aplicación.

La información de este tema solo es relevante si una aplicación se conecta a un gestor de colas IBM MQ o un bus de integración de servicios WebSphere Application Server. La información no es relevante para una conexión en tiempo real con un intermediario.

Para ejecutar transacciones locales, en primer lugar, una aplicación debe crear una sesión con transacción llamando al método Crear sesión de un objeto Connection, especificando como parámetro

que la sesión es una sesión con transacción. Por consiguiente, todos los mensajes enviados y recibidos dentro de la sesión se agrupan en una secuencia de transacciones. Una transacción finaliza cuando la aplicación confirma o retrotrae los mensajes que ha enviado y recibido desde que empezó la transacción.

Para confirmar una transacción, una aplicación llama al método Confirmar del objeto Session. Cuando se confirma una transacción, todos los mensajes enviados en la transacción pasan a estar disponibles para su entrega a otras aplicaciones, y todos los mensajes recibidos en la transacción reciben el acuse de recibo, de forma que el servidor de mensajería no los intenta volver a entregar a la aplicación. En el dominio punto a punto, el servidor de mensajería también elimina los mensajes recibidos de sus colas.

Para retrotraer una transacción, una aplicación llama al método Retrotraer del objeto Session. Cuando una transacción se retrotrae, el servidor de mensajería descarta todos los mensajes enviados en la transacción y todos los mensajes recibidos en la transacción pasan a estar disponibles para volverlos a entregar. En el dominio punto a punto, los mensajes que se han recibido se vuelven a colocar en sus colas y pasar a ser visibles de nuevo para otras aplicaciones.

Una nueva transacción se inicia automáticamente cuando una aplicación crea una sesión con transacción o llama al método Confirmar o Retrotraer. Por lo tanto, una sesión con transacción siempre tiene una transacción activa.

Cuando una aplicación cierra una sesión con transacción, se produce una retrotracción implícita. Cuando una aplicación cierra una conexión, se produce una retrotracción implícita para todas las sesiones con transacción de la conexión.

Una transacción está incluida íntegramente en una sesión con transacción. Una transacción no puede abarcar sesiones. Esto significa que no es posible para una aplicación enviar y recibir mensajes en dos o más sesiones con transacción y, después, confirmar o retrotraer todas estas acciones como una sola transacción.

Conceptos relacionados

Acuse de recibo de mensaje
Cada sesión que no es una sesión con transacción tiene una modalidad de acuse de recibo que determina cómo se acusa recibo de los mensajes recibidos por la aplicación. Hay disponibles tres modalidades de acuse de recibo y la selección de la modalidad de acuse de recibo afecta al diseño de la aplicación.

Entrega de mensajes
XMS admite las modalidades persistente y no persistente de la entrega de mensajes y la entrega asíncrona y síncrona de mensajes.

Acuse de recibo de mensaje

Cada sesión que no es una sesión con transacción tiene una modalidad de acuse de recibo que determina cómo se acusa recibo de los mensajes recibidos por la aplicación. Hay disponibles tres modalidades de acuse de recibo y la selección de la modalidad de acuse de recibo afecta al diseño de la aplicación.

La información de este tema solo es relevante si una aplicación se conecta a un gestor de colas IBM MQ o un bus de integración de servicios WebSphere Application Server. La información no es relevante para una conexión en tiempo real con un intermediario.

XMS utiliza el mismo mecanismo para acusar recibo de mensajes que utiliza JMS.

Si una sesión no es una sesión con transacción, la forma en la que se acusa recibo de los mensajes recibidos por la aplicación se determina mediante la modalidad de acuse de recibo de la sesión. Las tres modalidades de acuse de recibo se describen en los párrafos siguientes:

XMSC_AUTO_ACKNOWLEDGE

La sesión acusa recibo automáticamente de cada mensaje recibido por la aplicación.

Si los mensajes se entregan de forma síncrona a la aplicación, la sesión acusa recibo de cada mensaje cada vez que se completa una llamada Receive.

Si la aplicación recibe un mensaje correctamente, pero una anomalía impide el acuse de recibo, el mensaje pasa a estar disponible para volverse a entregar. Por lo tanto, la aplicación debe poder ser capaz de manejar un mensaje que se vuelve a entregar.

XMSC_DUPS_OK_ACKNOWLEDGE
La sesión acusa recibo de los mensajes recibidos por la aplicación en momentos que selecciona.

El uso de esta modalidad de acuse de recibo reduce la cantidad de trabajo que debe realizar la sesión, pero una anomalía que impide el acuse de recibo de mensaje podría provocar que más de un mensaje pasara a estar disponible para una nueva entrega. Por lo tanto, la aplicación debe poder ser capaz de manejar los mensajes que se vuelven a entregar.

XMSC_CLIENT_ACKNOWLEDGE
La aplicación acusa recibo de los mensajes que recibe llamando al método Acusar recibo de la clase Message.

La aplicación acusa recibo de cada mensaje de forma individual, o puede recibir un lote de mensajes y llamar al método Acusar recibo solo para el último mensaje que recibe. Cuando se llama al método Acusar recibo, se acusará recibo de todos los mensajes recibidos desde la última vez que se llamó al método.

Junto con cualquiera de estas modalidades de acuse de recibo, una aplicación puede detener y reiniciar la entrega de mensajes en una sesión llamando al método Recuperar de la clase Session. Se vuelven a entregar los mensajes de los que ya se había acusado recibo previamente. Sin embargo, podría ser que no se entregaran en la misma secuencia en la que se habían entregada anteriormente. Mientras tanto, podrían haber llegado los mensajes con prioridad superior y algunos de los mensajes originales podrían haber caducado. En el dominio punto a punto, algunos de los mensajes originales podrían haber sido consumidos por otra aplicación.

Una aplicación puede determinar si un mensaje se está volviendo a entregar examinando el contenido del campo de cabecera JMSRedelivered del mensaje. La aplicación lo hace llamando al método Obtener JMSRedelivered de la clase Message.

Conceptos relacionados

Sesiones con transacción
Las aplicaciones XMS pueden ejecutar transacciones locales. Una *transacción local* es una transacción que implica cambios solo en los recursos del gestor de colas o el bus de integración de servicios al cual está conectada la aplicación.

Entrega de mensajes
XMS admite las modalidades persistente y no persistente de la entrega de mensajes y la entrega asíncrona y síncrona de mensajes.

Entrega de mensajes

XMS admite las modalidades persistente y no persistente de la entrega de mensajes y la entrega asíncrona y síncrona de mensajes.

Modalidad de entrega de mensajes

XMS admite dos modalidades de entrega de mensajes:

Persistente
Los mensajes persistentes se entregan solo una vez. Un servidor de mensajería toma precauciones especiales como, por ejemplo, el registro de mensajes, para garantizar que los mensajes persistentes no se pierden en el tránsito, incluso en el caso de una anomalía.

No persistente
Los mensajes no persistentes se entregan más de una vez. Los mensajes no persistentes son menos fiables que los mensajes persistentes porque se pueden perder en el tránsito en caso de una anomalía.

La selección de la modalidad de entrega es una compensación entre la fiabilidad y el rendimiento. Normalmente, los mensajes no persistentes se transportan más rápidamente que los mensajes persistentes.

Entrega de mensajes asíncrona

XMS utiliza una hebra para manejar todas las entregas de mensajes asíncronas para una sesión. Esto significa que solo se puede ejecutar una función de escucha de mensajes o un método onMessage() a la vez.

Si más de un consumidor de mensajes de una sesión está recibiendo mensajes de forma asíncrona, y una función de escucha de mensajes o un método onMessage() está entregando un mensaje a un consumidor de mensajes, cualquier otro consumidor de mensajes que está esperando el mismo mensaje debe seguir esperando. Otros mensajes que están esperando a ser entregados en la sesión también deben seguir esperando.

Si una aplicación requiere una entrega simultánea de mensajes, cree más de una sesión de forma que XMS utilice más de una hebra para manejar la entrega de mensajes asíncrona. De esta forma, se pueden ejecutar de forma simultánea más de una función de escucha de mensajes o de un método onMessage().

Una sesión no se convierte en asíncrona asignado un escucha de mensajes a un consumidor. Una sesión se convierte en asíncrona solo cuando se llama al método Connection.Start. Todas las llamadas síncronas están permitidas hasta que se llama al método Connection.Start. La entrega de mensajes a consumidores se inicia cuando se llama a Connection.Start .

Si las llamadas síncronas, como la creación de un consumidor o productor, se deben realizar en una sesión asíncrona, se debe llamar a Connection.Stop . Una sesión se puede reanudar llamando al método Connection.Start para iniciar la entrega de mensajes. La única excepción a esto es la hebra de entrega de mensajes Session, que es la hebra que entrega mensajes a la función de devolución de llamada. Esta hebra puede realizar cualquier llamada en una sesión (excepto una llamada Close) en la función de devolución de llamada de mensaje.

Nota: En la modalidad no gestionada, la llamada MQDISC dentro de una función de devolución de llamada no está soportada por el cliente de IBM MQ .NET. De esta forma, la aplicación cliente no puede Crear o Cerrar sesiones en la devolución de llamada MessageListener en la modalidad de recepción Asíncrona. Cree y elimine la sesión fuera del método MessageListener.

Entrega de mensajes síncrona

Los mensajes se entregan de forma síncrona a una aplicación si la aplicación utiliza los métodos Recibir de objetos MessageConsumer.

Mediante los métodos Recibir, una aplicación puede esperar un mensaje un periodo de tiempo especificado, o puede esperar indefinidamente. De forma alternativa, si una aplicación no desea esperar un mensaje, puede utilizar el método Recibir sin espera.

Conceptos relacionados

Sesiones con transacción
Las aplicaciones XMS pueden ejecutar transacciones locales. Una *transacción local* es una transacción que implica cambios solo en los recursos del gestor de colas o el bus de integración de servicios al cual está conectada la aplicación.

Acuse de recibo de mensaje
Cada sesión que no es una sesión con transacción tiene una modalidad de acuse de recibo que determina cómo se acusa recibo de los mensajes recibidos por la aplicación. Hay disponibles tres modalidades de acuse de recibo y la selección de la modalidad de acuse de recibo afecta al diseño de la aplicación.

Destinos

Una aplicación XMS utiliza un objeto Destination para especificar el destino de mensajes que se están enviando y el origen de mensajes que se están recibiendo.

Una aplicación XMS puede crear un objeto Destination durante el tiempo de ejecución, u obtener un destino predefinido del repositorio de objetos administrados.

Al igual que con ConnectionFactory, la forma más flexible para que una aplicación XMS especifique un destino es definirlo como un objeto administrado. Utilizando este enfoque, las aplicaciones escritas enlenguajes C, C++ y .NET y Java pueden compartir definiciones del destino. Las propiedades de objetos Destination administrados se pueden cambiar sin cambiar ningún código.

Para aplicaciones .NET, cree un destino utilizando el método CreateTopic o CreateQueue. Estos dos métodos están disponibles en ambos objetos, ISession y XMSFactoryFactory, en la API .NET. Para obtener más información, consulte “Destinos en .NET” en la página 671 y ../refdev/sapidest.dita#sapidest.

Identificadores uniformes de recursos de tema

El identificador uniforme de recursos (URI) de tema especifica el nombre del tema; también puede especificar una o más propiedades para el mismo.

El URI para un tema empieza con la secuencia topic://, seguida del nombre del tema y (opcional) una lista de pares de nombre-valor que establecen las propiedades de tema restantes. Un nombre de tema no puede estar vacío.

A continuación, aparece un ejemplo en un fragmento de código de .NET:

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Para obtener más información sobre las propiedades de un tema, incluyendo el nombre y los valores válidos que puede utilizar en un URI, consulte Propiedades de destino.

Al especificar un URI de tema para su uso en una suscripción, se pueden utilizar los comodines. La sintaxis de estos comodines depende del tipo de conexión y de la versión del intermediario; está disponible la siguiente opción:

- Bus de integración de servicios WebSphere Application Server

Bus de integración de servicios WebSphere Application Server

El bus de integración de servicios WebSphere Application Server utiliza los caracteres comodín siguientes:

* para coincidir con cualquier carácter en un nivel de jerarquía
// para coincidir con 0 o más niveles
//. para coincidir con 0 o más niveles (al final de una expresión de tema)

Tabla 82 en la página 658 proporciona algunos ejemplos sobre cómo utilizar este esquema de comodín.

Tabla 82. URI de ejemplo que utilizan el esquema de comodín para el bus de integración de servicios WebSphere Application Server

Identificador uniforme de recursos	Coincide con	Ejemplos
"topic://Sport/*ball/Results"	Todos los temas con un nombre de nivel jerárquico único que acaban con"ball" entre Sport y Results	"topic://Sport/Football/Results" y "topic://Sport/Netball/Results"
"topic://Sport//Results"	Todos los temas que empiezan con "Sport/" y acaban en "/Results"	"topic://Sport/Football/Results" y "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	Todos los temas que empiezan con "Sport/Football/"	"topic://Sport/Football/Results" y "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/*ball//Results//."	Temas	"topic://Sport/Football/Results" y "topic://Sport/Netball/National/Div3/Results/2002/November"

Conceptos relacionados

Identificadores uniformes de recursos de cola
El URI de una cola especifica el nombre de la cola; también puede especificar una o más propiedades de la cola.

Destinos temporales
Las aplicaciones XMS pueden crear y utilizar destinos temporales.

Identificadores uniformes de recursos de cola

El URI de una cola especifica el nombre de la cola; también puede especificar una o más propiedades de la cola.

El URI de una cola empieza con la secuencia `queue://`, seguida por el nombre de la cola; también podría incluir una lista de pares de nombre-valor que establecen las propiedades de cola restantes.

Para colas de IBM MQ (pero no para colas del proveedor de mensajería predeterminado WebSphere Application Server), el gestor de colas en el cual reside la cola se puede especificar antes de la cola, con una / que separa el nombre del gestor de colas del nombre de cola.

Si se especifica un gestor de colas, debe ser uno al que XMS está conectado directamente para la conexión utilizando esta cola, o debe ser accesible desde esta cola. Los gestores de colas remotos solo están soportados para recuperar mensajes de colas, no para colocar mensajes en colas. Si desea detalles completos, consulte la documentación del gestor de colas IBM MQ.

Si no se especifica ningún gestor de colas, el separador/ adicional es opcional, y su presencia o ausencia es indiferente para la definición de la cola.

Las siguientes definiciones de cola son todas equivalentes para una cola IBM MQ denominada QB en un gestor de colas denominado QM_A, al que XMS está conectado directamente:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Conceptos relacionados

Identificadores uniformes de recursos de tema
El identificador uniforme de recursos (URI) de tema especifica el nombre del tema; también puede especificar una o más propiedades para el mismo.

Destinos temporales
Las aplicaciones XMS pueden crear y utilizar destinos temporales.

Destinos temporales

Las aplicaciones XMS pueden crear y utilizar destinos temporales.

Normalmente, una aplicación utiliza un destino temporal para recibir respuestas a mensajes de solicitud. Para especificar el destino adónde se va a enviar un mensaje de solicitud, una aplicación llama al método Establecer JMSReplyTo del objeto Message que representa el mensaje de solicitud. El destino especificado en la llamada puede ser un destino temporal.

Aunque se utiliza una sesión para crear un destino temporal, el ámbito de un destino temporal es, realmente, la conexión que se había utilizado para crear la sesión. Cualquiera de las sesiones de la conexión puede crear productores de mensajes y consumidores de mensajes para el destino temporal. El destino temporal perdura, hasta que se suprime de forma explícita, o finaliza la conexión, lo que se produzca antes.

Cuando una aplicación crea una cola temporal, se crea una cola en el servidor de mensajería al que se conecta la aplicación. Si la aplicación está conectada a un gestor de colas, se crea una cola dinámica a partir de la cola modelo cuyo nombre se especifica mediante la propiedad XMSC_WMQ_TEMPORARY_MODEL , y el prefijo que se utiliza para formar el nombre de la cola dinámica se especifica mediante la propiedad XMSC_WMQ_TEMP_Q_PREFIX . Si la aplicación está conectada a un bus de integración de servicios, se crea una cola temporal en el bus y el prefijo que se utiliza para formar el nombre de la cola temporal se especifica mediante la propiedad XMSC_WPM_TEMP_Q_PREFIX .

Cuando una aplicación que está conectada a un bus de integración de servicios crea un tema temporal, el prefijo que se utiliza para formar el nombre del tema temporal se especifica mediante la propiedad XMSC_WPM_TEMP_TOPIC_PREFIX .

Conceptos relacionados

Identificadores uniformes de recursos de tema
El identificador uniforme de recursos (URI) de tema especifica el nombre del tema; también puede especificar una o más propiedades para el mismo.

Identificadores uniformes de recursos de cola
El URI de una cola especifica el nombre de la cola; también puede especificar una o más propiedades de la cola.

Productores de mensajes

En XMS, se puede crear un productor de mensajes ya sea con un destino válido o sin ningún destino asociado. Al crear un productor de mensajes con un destino nulo, se debe especificar un destino válido al enviar un mensaje.

Productores de mensajes con destino asociado

En este escenario, el productor de mensajes se crea utilizando un destino válido. Durante la operación de envío, no es necesario especificar el destino.

Productores de mensajes sin destino asociado

En XMS .NET, se puede crear un productor de mensajes con un destino nulo.

Para crear un productor de mensajes sin destino asociado cuando se utiliza la API .NET , se debe pasar NULL como parámetro en el método `CreateProducer()` del objeto ISession (por ejemplo, `session.CreateProducer(null)`). Sin embargo, debe especificarse un destino válido cuando se envía el mensaje.

Consumidores de mensajes

Los consumidores de mensajes se pueden clasificar como suscriptores duraderos y suscriptores no duraderos y consumidores de mensajes síncronos y mensajes asíncronos.

Suscriptores duraderos

Un suscriptor duradero es un consumidor de mensajes que recibe todos los mensajes publicados sobre un tema, incluyendo los mensajes publicados mientras el suscriptor está inactivo.

La información de este tema solo es relevante si una aplicación se conecta a un gestor de colas IBM MQ o un bus de integración de servicios WebSphere Application Server. La información no es relevante para una conexión en tiempo real con un intermediario.

Para crear un suscriptor duradero para un tema, una aplicación llama al método Crear suscriptor duradero de un objeto Session, especificando como parámetros un nombre que identifica la suscripción duradera y un objeto Destination que representa el tema. La aplicación puede crear un suscriptor duradero con o sin un selector de mensajes, y puede especificar si el suscriptor duradero va a recibir mensajes publicados por su propia conexión.

La sesión utilizada para crear un suscriptor duradero debe tener un identificador de cliente asociado. El identificador de cliente es el mismo que el asociado a la conexión que se utiliza para crear la sesión; se especifica como se describe en “Objetos ConnectionFactories y Connection” en la página 652.

El nombre que identifica la suscripción duradera debe ser exclusivo dentro del identificador de cliente y, por lo tanto, el identificador de cliente forma parte del identificador único completo de la suscripción duradera. El servidor de mensajería mantiene un registro de la suscripción duradera y garantiza que se conservan todos los mensajes publicados sobre el tema, hasta que reciban un acuse de recibo del suscriptor duradero o hasta que caduquen.

El servidor de mensajería sigue manteniendo el registro de la suscripción duradera, incluso después de que se cierre el suscriptor duradero. Para reutilizar una suscripción duradera que se ha creado anteriormente, una aplicación debe crear un suscriptor duradero especificando el mismo nombre de suscripción, y utilizando una sesión con el mismo identificador de cliente, que los asociados a la suscripción duradera. Solo una sesión a la vez puede tener un suscriptor duradero para una suscripción duradera concreta.

El ámbito de una suscripción duradera es el servidor de mensajería que mantiene un registro de la suscripción. Si dos aplicaciones conectadas a distintos servidores de mensajería crean cada una un suscriptor duradero utilizando el mismo nombre de suscripción e identificador de cliente, se crean dos suscripciones duraderas completamente independientes.

Para suprimir una suscripción duradera, una aplicación llama al método Anular suscripción de un objeto Session, especificando como parámetro el nombre que identifica la suscripción duradera. El identificador de cliente asociado a la sesión debe ser el mismo que está asociado a la suscripción duradera. El servidor de mensajería suprime el registro de la suscripción duradera que está manteniendo y no envía ningún mensaje más al suscriptor duradero.

Para cambiar una suscripción existente, una aplicación puede crear un suscriptor duradero utilizando el mismo nombre de suscripción e identificador de cliente, pero especificando un tema o un selector de mensajes diferente (o ambos). El cambio de una suscripción duradera es equivalente a suprimir la suscripción y crear una nueva.

Para una aplicación que se conecta a un gestor de colas de IBM MQ , XMS gestiona las colas de suscriptores. De ahí que no es necesario que la aplicación especifique una cola de suscriptor. XMS ignorará la cola de suscriptor, si se ha especificado.

Tenga en cuenta que no puede cambiar la cola de suscriptor para una suscripción duradera. La única forma de cambiar la cola de suscriptor es suprimir la suscripción y crear una nueva.

Para una aplicación que se conecta a un bus de integración de servicios, cada suscriptor duradero debe tener un inicio de suscripción duradera designada. Para especificar el inicio de la suscripción duradera para todos los suscriptores duraderos que utilizan la misma conexión, establezca la propiedad XMSC_WPM_DUR_SUB_HOME del objeto ConnectionFactory que se utiliza para crear la conexión. Para especificar el inicio de la suscripción duradera para un tema individual, establezca la propiedad XMSC_WPM_DUR_SUB_HOME del objeto Destination que representa el tema. Se debe especificar un inicio de suscripción duradera para una conexión antes de que una aplicación pueda crear una suscriptor duradero que utilice la conexión. Cualquier valor especificado para un destino altera temporalmente el valor especificado para la conexión.

Consumidores de mensajes síncronos y asíncronos

El consumidor de mensajes síncronos recibe los mensajes de una cola de forma síncrona y el consumidor de mensajes asíncronos recibe el mensaje de una cola de forma asíncrona.

Consumidores de mensajes síncronos

Un consumidor de mensajes síncrono recibe un mensaje cada vez. Cuando se utiliza el método `Receive(wait interval)` , la llamada sólo espera un periodo de tiempo especificado en milisegundos para un mensaje o hasta que se cierra el consumidor de mensajes.

Si se utiliza el método ReceiveNoWait(), el consumidor de mensajes síncrono recibe mensajes sin ningún retardo; si está disponible el siguiente mensaje, se recibe inmediatamente, de lo contrario, se devuelve un puntero de un objeto Message nulo.

Consumidores de mensajes asíncronos

El escucha de mensajes registrado por la aplicación se invoca siempre que está disponible un nuevo mensaje en la cola.

Mensajes dañados en XMS

Un mensaje dañado es un mensaje que no puede procesar una aplicación MDB receptora. Si se encuentra un mensaje dañado, el objeto XMS MessageConsumer puede volverlo a poner en cola de acuerdo con dos propiedades de cola, BOQUEUE y BOTHRESH.

En algunas circunstancias, un mensaje entregado a un MDB se podría retrotraer en una cola IBM MQ. Por ejemplo, esto puede suceder si se entrega un mensaje en una unidad de trabajo que, posteriormente, se retrotrae. Un mensaje que se retrotrae, normalmente, se vuelve a entregar, pero un mensaje con un formato incorrecto podría provocar de forma repetida que falle un MDB y, por lo tanto, no se puede entregar. Dicho mensaje se denomina mensaje dañado. Puede configurar IBM MQ de forma que el mensaje dañado se transfiere automáticamente a otra cola para una investigación adicional o para descartarse. Si desea más información sobre cómo configurar IBM MQ de esta forma, consulte Manejo de mensajes dañados en ASF.

A veces, a una cola llega un mensaje con un formato incorrecto. En este contexto, un formato incorrecto significa que la aplicación receptora no puede procesar el mensaje correctamente. Dicho mensaje puede provocar que la aplicación receptora falle y restituya este mensaje con formato incorrecto. El mensaje se puede entregar repetidamente a la cola de entrada y la aplicación lo puede restituir también repetidamente. Estos mensajes son conocidos como mensajes dañados. El objeto XMS MessageConsumer detecta mensajes dañados y los redirecciona a un destino alternativo.

El gestor de colas IBM MQ conserva un registro del número de veces que se ha restituido cada mensaje. Cuando este número alcanza un valor de umbral configurable, el consumidor de mensajes vuelve a poner en cola al mensaje en una cola de retirada especificada. Si esta recolocación en cola falla por cualquier motivo, el mensaje se elimina de la cola de entrada y se vuelve a poner en cola en la cola de mensajes no entregados, o se descarta.

Los objetos XMS ConnectionConsumer manejan mensajes dañados de la misma forma y utilizan las mismas propiedades de cola. Si varios consumidores de conexiones están supervisando la misma cola, es posible que el mensaje dañado se pueda entregar a una aplicación más veces que el valor de umbral, antes de que se produzca la recolocación en cola. Este comportamiento se debe a la forma en la que los consumidores de conexionesindividuales supervisan las colas y vuelven a colocar en cola mensajes dañados.

El valor umbral y el nombre de la cola de retirada son atributos de una cola de IBM MQ. Los nombres de los atributos son BackoutThreshold y BackoutRequeueQName. La cola a la que se aplican es la siguiente:

- Para la mensajería punto a punto, esta es la cola local subyacente. Esto es importante cuando los consumidores de mensajes y los consumidores de conexiones utilizan alias de cola.
- Para la mensajería de publicación/suscripción en la modalidad normal de proveedor de mensajería IBM MQ, esta es la cola modelo a partir de la que se ha creado la cola gestionada del objeto Tema.
- Para la mensajería de publicación/suscripción en la modalidad de migración del proveedor de mensajería IBM MQ, esta es la cola CCSUB definida en el objeto TopicConnectionFactory, o la cola CCDSUB definida en el objeto Topic.

Para establecer los atributos BackoutThreshold y BackoutRequeueQName, emita el mandato MQSC siguiente:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Para la mensajería de publicación/suscripción, si el sistema crea una cola dinámica para cada suscripción, estos valores de atributo se obtienen de la cola modelo de IBM MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Para modificar estos valores, utilice:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Si el valor del umbral de restitución es cero, el manejo de mensajes dañados está inhabilitado, y los mensajes dañados permanecen en la cola de entrada. De lo contrario, cuando el recuento de restitución alcanza el valor de umbral, el mensaje se envía a la cola de retirada especificada.

Si el recuento de restitución alcanza el valor de umbral, pero el mensaje no puede ir a la cola de retirada, el mensaje se envía a la cola de mensajes no entregados o, si el mensaje es no persistente, se descarta.

Esta situación se produce si la cola de retirada no está definida, o si el objeto MessageConsumer no puede enviar el mensaje a la cola de retirada.

Configuración del sistema para realizar el manejo de mensajes con formato incorrecto

La cola que XMS .NET utiliza al consultar los atributos **BOTHRESH** y **BOQNAME** depende del estilo de mensajería que se esté realizando:

- Para la mensajería punto a punto, esta es la cola local subyacente. Esto es importante cuando una aplicación XMS .NET está consumiendo mensajes de colas alias o colas de clúster.
- Para la mensajería de publicación/suscripción, se crea una cola gestionada para contener los mensajes para una aplicación. XMS .NET consulta la cola gestionada para determinar los valores de los atributos **BOTHRESH** y **BOQNAME** .

  La cola gestionada se crea a partir de una cola de modelo asociada al objeto de tema al que está suscrita la aplicación y hereda los valores de los atributos **BOTHRESH** y **BOQNAME** de la cola de modelo. La cola de modelo que se utiliza depende de si la aplicación receptora ha extraído una suscripción duradera o no duradera:

  – La cola de modelo utilizada para las suscripciones duraderas se especifica mediante el atributo **MDURMDL** del tema. El valor predeterminado de este atributo es `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Para las suscripciones no duraderas, la cola de modelo que se utiliza se especifica mediante el atributo **MNDURMDL**. El valor predeterminado del atributo **MNDURMDL** es `SYSTEM.NDURABLE.MODEL.QUEUE`.

Al consultar los atributos **BOTHRESH** y **BOQNAME** , XMS .NET:

- Abre la cola local o la cola de destino para una cola alias.
- Consulta los atributos **BOTHRESH** y **BOQNAME** .
- Cierra la cola local o la cola de destino para una cola alias.

Las opciones abiertas que se utilizan cuando se abre una cola local, o la cola de destino para una cola alias, dependen de la versión de IBM MQ que se está utilizando:

- Para IBM MQ 9.1.0 Fix Pack 4 Long Term Support y anteriores, y IBM MQ 9.1.4 Continuous Delivery y anteriores: si la cola local, o la cola de destino para una cola alias, es una cola de clúster, XMS .NET abre la cola con las opciones `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` y `MQOO_FAIL_IF_QUIESCING` . Esto significa que el usuario que ejecuta la aplicación receptora debe tener acceso de consulta y obtención en la instancia local de la cola del clúster.

  XMS .NET abre todos los otros tipos de colas locales con las opciones abiertas `MQOO_INQUIRE` y `MQOO_FAIL_IF_QUIESCING`. Para que XMS .NET consulte los valores de los atributos, el usuario que ejecuta la aplicación receptora debe tener acceso de consulta en la cola local.
- Cuando se utiliza XMS .NET desde IBM MQ 9.1.5 y IBM MQ 9.1.0 Fix Pack 5, el usuario que ejecuta la aplicación receptora debe tener acceso de consulta en la cola local, independientemente del tipo de cola.

Para mover mensajes con formato incorrecto a una cola de reposición en cola de restitución o a la cola de mensajes no entregados del gestor de colas, debe otorgar al usuario que ejecuta la aplicación las autorizaciones `put` y `passall` .

Manejo de mensajes dañados en ASF
Al utilizar los Recursos del servidor de aplicaciones (ASF), el ConnectionConsumer, y no MessageConsumer, procesa mensajes dañados. El ConnectionConsumer vuelve a colocar en cola mensajes de acuerdo con las propiedades BackoutThreshold y BackoutRequeueQName de la cola.

Cuando una aplicación utiliza ConnectionConsumers, las circunstancias en las cuales se restituye un mensaje dependen de la sesión que proporciona el servidor de aplicaciones:

- Cuando la sesión es una sesión sin transacción, con AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE, un mensaje solo se restituye después de un error del sistema, o si la aplicación termina de forma inesperada
- Cuando la sesión es una sesión sin transacción con CLIENT_ACKNOWLEDGE, los mensajes sin acuse de recibo se pueden restituir mediante el servidor de aplicaciones que llama `Session.recover()`.

  Normalmente, la implementación de cliente de MessageListener o el servidor de aplicaciones llamando a `Message.acknowledge()`. `Message.acknowledge()` reconoce todos los mensajes entregados en la sesión, hasta el momento.
- Cuando la sesión es una sesión con transacción, los mensajes no reconocidos se puede restituir mediante el servidor de aplicaciones llamando a `Session.rollback()`.

Examinadores de colas

Una aplicación utiliza un examinador de colas para examinar mensajes en una cola sin eliminarlas.

Para crear un examinador de colas, una aplicación llama al método Crear examinador de colas de un objeto ISession, especificando como parámetro un objeto Destination que identifica la cola que se va a examinar. La aplicación puede crear un examinador de colas con o sin un selector de mensajes.

Tras crear un examinador de colas, la aplicación puede llamar al método GetEnumerator del objeto IQueueBrowser para obtener una lista de los mensajes en la cola. El método devuelve un enumerador que encapsula una lista de objetos Message. El orden de los objetos Message de la lista es el mismo que el orden en el cual los mensajes deberían recuperarse de la cola. La aplicación puede utilizar el enumerador para examinar cada mensaje a su vez.

El enumerador se actualiza de forma dinámica a medida que los mensajes se colocan en la cola y se eliminan de la cola. Cada vez que la aplicación llama a IEnumerator.MoveNext() para examinar el siguiente mensaje en la cola, el mensaje refleja el contenido actual de la cola.

Una aplicación puede llamar al método GetEnumerator más de una vez para un examinador de colas determinado. Cada llamada devuelve un nuevo enumerador. Por lo tanto, la aplicación puede utilizar más de un enumerador para examinar los mensajes de una cola y mantener varias posiciones dentro de la cola.

Una aplicación puede utilizar un examinador de colas para buscar un mensaje adecuado para eliminarlo de una cola y, después, utilizar un consumidor de mensajes con un selector de mensajes para eliminar el mensaje. El selector de mensajes puede seleccionar el mensaje de acuerdo con el valor del campo de cabecera JMSMessageID. Si desea más información sobre este y otros campos de cabecera de mensaje JMS, consulte “Campos de cabecera en un mensaje XMS” en la página 686.

Solicitantes

Una aplicación utiliza un solicitante para enviar un mensaje de solicitud y, después, esperar y recibir la respuesta.

Muchas aplicaciones de mensajería se basan en algoritmos que envían un mensaje de solicitud y, después, esperan una respuesta. XMS proporciona una clase llamada Requestor para ayudar con el desarrollo de este estilo de aplicación.

Para crear un solicitante, una aplicación llama al constructor Create Requestor de la clase Requestor, especificando como parámetros un objeto Session y un objeto Destination que identifica dónde se van a enviar los mensajes de solicitud. La sesión no debe ser una sesión con transacción ni debe tener una

modalidad de acuse de recibo de XMSC_CLIENT_ACKNOWLEDGE. El constructor crea automáticamente una cola o tema temporal donde se van a enviar los mensajes de respuesta.

Después de crear un solicitante, la aplicación puede llamar al método Solicitar del objetoRequestor para enviar un mensaje de solicitud y, después, esperar una respuesta y recibirla de la aplicación que recibe el mensaje de solicitud. La llamada espera hasta que se recibe la respuesta o hasta que finaliza la sesión, lo que se produzca antes. El solicitante solo necesita una respuesta para cada mensaje de solicitud.

Cuando la aplicación cierra el solicitante, la cola o tema temporal se suprime. Sin embargo, la sesión asociada no se cierra.

Supresión de objeto

Cuando una aplicación suprime un objeto XMS que ha creado, XMS libera los recursos internos que se han asignado al objeto.

Cuando una aplicación crea un objeto XMS, XMS asigna memoria y otros recursos al objeto. XMS conserva estos recursos internos hasta que la aplicación suprime de forma explícita el objeto llamando al método cerrar o suprimir del objeto, en dicho punto XMS libera los recursos internos. Si una aplicación intenta suprimir un objeto que ya se ha suprimido, se ignora la llamada.

Cuando una aplicación suprime un objeto Connection o Session, XMS suprime determinados objetos asociados automáticamente y libera sus recursos internos. Estos son objetos que han sido creados por el objeto Connection o Session y no tienen ninguna función independiente del objeto. Estos objetos se muestra en Tabla 83 en la página 665.

Nota: Si una aplicación cierra una conexión con sesiones dependientes, todos los objetos que dependen de estas sesiones también se suprimen. Solo un objeto Connection o Session puede tener objetos dependientes.

Tabla 83. Objetos que se suprimen automáticamente

Objeto suprimido	**Método**	**Objetos dependientes que se suprimen automáticamente**
Conexión	Cerrar conexión	Objetos ConnectionMetaData y Session
Session	Cerrar sesión	Objetos MessageConsumer, MessageProducer, QueueBrowser y Requestor

Transacciones XA IBM MQ gestionadas a través de XMS

Las transacciones XA IBM MQ gestionadas se pueden utilizar a través de XMS.

Para utilizar transacciones XA a través de XMS, se debe crear una sección con transacción. Cuando la transacción XA se está utilizando, el control de transacciones se realiza a través de transacciones globales DTC (Distributed Transaction Coordinator) y no a través de sesiones XMS. Cuando se utilizan transacciones XA, no se puede emitir `Session.commit` o `Session.rollback` en la sesión XMS. En su lugar, utilice los métodos DTC `Transscope.Commit` o `Transscope.Rollback` para confirmar o retrotraer las transacciones. Si se utiliza una sesión para transacciones XA, el productor o consumidor que se crea utilizando la sesión debe formar parte de la transacción XA. No se pueden utilizar para ninguna operación fuera del ámbito de transacciones XA. No se pueden utilizar para operaciones como `Producer.send` o `Consumer.receive` fuera de la transacción XA.

Se lanza un objeto de excepción `IllegalStateException` si:

- Se utiliza una sesión con transacción XA para `Session.commit` o `Session.rollback`.
- Los objetos de productor o consumidor que se han utilizado una vez en una sesión con transacción XA se utilizan fuera del ámbito de transacciones XA.

Las transacciones XA no están soportadas en consumidores asíncronos.

Nota:

1. Se podría emitir un cierre en el objeto `Producer`, `Consumer`, `Session` o `Connection` antes de confirmar la transacción XA. En cuyo caso, los mensajes de la transacción se retrotraen. De forma similar, si la conexión se interrumpe antes de confirmar la transacción XA, se retrotraen todos los mensajes de la transacción. Para un objeto `Producer`, una retrotracción significa que los mensajes no se colocan en la cola. Para un objeto `Consumer`, una retrotracción significa que los mensajes permanecen en la cola.
2. Si un objeto `Producer` coloca un mensaje con `TimeToLive` en `TransactionScope` y se emite `commit` una vez que ha transcurrido el tiempo, el mensaje puede caducar antes de que se emita `commit`. En este caso, el mensaje no se pondrá a disposición de los objetos `Consumer`.
3. Los objetos `Session` no están soportados entre las hebras. No está soportado el uso de transacciones con los objetos `Session` que se comparten entre hebras.

Tipos primitivos de XMS

XMS proporciona equivalentes de los ochos tipos primitivos de Java (byte, short, int, long, float, double, char y boolean). Esto permite el intercambio de mensajes entre XMS y JMS sin perder o dañar datos.

Tabla 84 en la página 666 lista el tipo de datos, tamaño y valor mínimo y máximo equivalentes de Java de cada tipo primitivo de XMS.

Tabla 84. Tipos de datos de XMS y sus equivalentes Java

Tipo de datos XMS	Tipo de datos Java compatible	Tamaño	Valor mínimo	Valor máximo
System.Boolean	boolean	32 bits	falso	true
System.SBYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.Int16	short	16 bits	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 bits	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	Entero largo	64 bits	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	float	32 bits	-3.402823E-38 (hasta 7-dígitos de precisión)	3.402823E+38 (hasta 7-dígitos de precisión)
System.Double	double	64 bits	-1.79769313486231E-308 (hasta 15-dígitos de precisión)	1.79769313486231E+308 (hasta 15-dígitos de precisión)

Conversión implícita de un valor de propiedad de un tipo de datos a otro

Cuando una aplicación obtiene el valor de una propiedad, el valor se puede convertir mediante XMS a otro tipo de datos. Muchas normas rigen qué conversiones están soportadas y cómo XMS realiza las conversiones.

Una propiedad de un objeto que tiene un nombre y un valor; el valor tiene un tipo de datos asociado, donde también se hace referencia al valor de una propiedad como el *tipo de propiedad*.

Una aplicación utiliza los métodos de la clase PropertyContext para obtener y establecer las propiedades de objetos. Para poder obtener el valor de una propiedad, una aplicación llama al método que es apropiado para el tipo de propiedad. Por ejemplo, para obtener el valor de una propiedad de entero, normalmente, una aplicación llama al método GetIntProperty.

Sin embargo, cuando una aplicación obtiene el valor de una propiedad, XMS puede convertir el valor a otro tipo de datos. Por ejemplo, para obtener el valor de una propiedad de entero, una aplicación

puede llamar al método GetStringProperty, que devuelve el valor de la propiedad como una serie. Las conversiones soportadas por XMS se muestran en Tabla 85 en la página 667.

Tabla 85. Conversiones soportadas de un tipo de propiedad a otros tipos de datos

Tipo de propiedad	Tipos de datos de destino soportados
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String
System.Int16	System.String, System.Int32, System.Int64

Las reglas generales siguientes rigen las conversiones soportadas:

- Los valores de propiedad numéricos se pueden convertir de un tipo de datos a otro, siempre que no se pierda ningún dato durante la conversión. Por ejemplo, el valor de una propiedad con el tipo de datos System.Int32 se puede convertir a un valor con el tipo de datos System.Int64, pero no se puede convertir a un valor con el tipo de datos System.Int16.
- Un valor de propiedad de cualquier tipo de datos se puede convertir a una serie.
- Un valor de propiedad de serie se puede convertir a cualquier otro tipo de datos, siempre que la serie tenga el formato correcto para la conversión. Si una aplicación intenta convertir un valor de propiedad de serie que no tiene un formato correcto, XMS puede devolver errores.
- Si una aplicación intenta una conversión que no está soportada, XMS puede devolver un error.

Las reglas siguientes se aplican cuando un valor de propiedad se convierte de un tipo de datos a otro:

- Al convertir una propiedad booleana a una serie, el valor verdadero se convierte a la serie "true", y el valor falso se convierte a la serie "false".
- Al convertir un valor de propiedad booleana a un tipo de datos numérico, incluido System.SByte, el valor verdadero se convierte a 1, y el valor falso se convierte a 0.
- Al convertir un valor de propiedad de serie a un valor booleano, la serie "true" (no distingue entre mayúsculas y minúsculas), o "1" se convierte a verdadero y la serie "false" (no distingue entre mayúsculas y minúsculas) o "0" se convierte a falso. Todas las demás series no se pueden convertir.
- Al convertir un valor de propiedad de serie a un valor con el tipo de datosSystem.Int32, System.Int64, System.SByte o System.Int16, la serie debe tener el formato siguiente.

  [*espacios en blanco*][*signo*]*dígitos*

  Los componentes de serie se definen del modo siguiente:

  espacios en blanco
  Caracteres en blanco iniciales opcionales.

  signo
  Un carácter opcional de signo más (+) o signo menos (-).

  dígitos
  Una secuencia continua de caracteres de dígitos (0-9). Al menos, debe estar presente un carácter de dígito.

Después de la secuencia de caracteres de dígitos, la serie puede contener otros caracteres que no son caracteres de dígitos, pero la conversión se detiene tan pronto como se llega al primero de estos caracteres. Se da por sentado que la serie representa un entero decimal.

XMS puede devolver un error si la serie no tiene un formato correcto.

- Al convertir un valor de propiedad de serie a un valor con el tipo de datosSystem.Double o System.Float, la serie debe tener el formato siguiente:

  [*espacios en blanco*][*signo*][*dígitos*][*punto*[*d_dígitos*]][*e_car*[*e_signo*]*e_dígitos*]

  Los componentes de serie se definen del modo siguiente:

  espacios en blanco
  (Opcional) Caracteres en blanco iniciales.

  signo
  (Opcional) Carácter de signo más (+) o signo menos (-).

  dígitos
  Una secuencia continua de caracteres de dígitos (0-9). Al menos, debe estar presente un carácter de un dígito en *dígitos* o *d_dígitos*.

  punto
  (Opcional) Punto decimal (.).

  d_dígitos
  Una secuencia continua de caracteres de dígitos (0-9). Al menos, debe estar presente un carácter de un dígito en *dígitos* o *d_dígitos*.

  e_car
  Un carácter exponente, que puede ser *E* o *e*.

  e_signo
  (Opcional) Carácter de signo más (+) o signo menos (-) para el exponente.

  e_dígitos
  Una secuencia contigua de caracteres de dígitos (0-9) para el exponente. Al menos, debe estar presente un carácter de dígito, si la serie contiene un carácter de exponente.

  Detrás de la secuencia de caracteres de dígitos, o los caracteres opcionales que representan un exponente, la serie puede contener otros caracteres que no son caracteres de dígitos, pero la conversión se detiene tan pronto como se llega al primero de estos caracteres. Se da por supuesto que la serie representa un número de coma flotante decimal con un exponente que es una potencia de 10.

  XMS puede devolver un error si la serie no tiene un formato correcto.

- Al convertir un valor de propiedad numérica a una serie, incluyendo un valor de propiedad con el tipo de datos System.SByte, el valor se convierte a la representación de serie del valor como número decimal, no la serie que contiene el carácter ASCII para dicho valor. Por ejemplo, el entero 65 se convierte a la serie "65", no la serie "A".
- Al convertir un valor de propiedad de matriz de bytes a una serie, cada byte se convierte a los 2 caracteres hexadecimales que representan el byte. Por ejemplo, la matriz de bytes {0xF1, 0x12, 0x00, 0xFF} se convierte a la serie "F11200FF".

Las conversiones de un tipo de propiedad a otros tipos de datos están soportadas por los métodos de ambas clases, Property y PropertyContext.

Iteradores

Un iterador encapsula una lista de objetos y un cursor que mantiene la posición actual en la lista. El concepto de Iterador, tal como está disponible en IBM MQ Message Service Client (XMS) for C/C++, se implementa utilizando la interfaz IEnumerator en IBM MQ Message Service Client (XMS) for .NET.

Cuando se crea un iterador, la posición del cursor se encuentra antes del primer objeto. Una aplicación utiliza un iterador para recuperar cada objeto a su vez.

La clase Iterator de IBM MQ Message Service Client (XMS) for C/C++ es equivalente a la clase Enumerator en Java. IBM MQ Message Service Client (XMS) for .NET es similar a Java y utiliza una interfaz IEnumerator.

Una aplicación puede utilizar un IEnumerator para realizar las tareas siguientes:

- Para obtener las propiedades de un mensaje
- Para obtener los pares de nombre-valor del cuerpo de un mensaje de correlación
- Para examinar los mensajes de una cola
- Para obtener los nombres de las propiedades de mensaje definidas por JMS soportadas por una conexión

Identificadores de juego de caracteres codificado

En XMS .NET, todas las series se pasan utilizando la serie .NET nativa. Puesto que esto tiene una codificación fija, no es necesaria más información para interpretarla. Por lo tanto, la propiedad XMSC_CLIENT_CCSID no es necesaria para las aplicaciones XMS .NET.

Códigos de error y excepción de XMS

XMS utiliza un rango de códigos de error para indicar anomalías. Estos códigos de error no se listan de forma explícita en esta documentación porque pueden variar de release a release. Solo se documentan los códigos de excepción de XMS (con el formato XMS_X_...) porque permanecen igual entre los mismos releases de XMS.

Reconexión automática del cliente IBM MQ a través de XMS

Configure el cliente XMS para reconectarse automáticamente después de una anomalía de red, de gestor de colas o de servidor mientras utiliza el cliente IBM WebSphere MQ 7.1 y posterior como proveedor de mensajes.

Utilice las propiedades `WMQ_CONNECTION_NAME_LIST` y `WMQ_CLIENT_RECONNECT_OPTIONS` de la clase `MQConnectionFactory` para configurar una conexión de cliente para que se reconecte automáticamente. La reconexión automática de cliente reconecta un cliente después de una anomalía de conexión, o como opción después de detener el gestor de colas. El diseño de algunas aplicaciones cliente las invalida para la reconexión automática.

Las conexiones de cliente reconectable de forma automática pasan a ser reconectables una vez que se ha establecido la conexión.

Nota: Las propiedades `Opciones de reconexión de cliente`, `Tiempo de espera de reconexión de cliente` y `Lista de nombres de conexión` también se pueden establecer a través de la Tabla de definiciones de canal de cliente (CCDT) o habilitando la reconexión de cliente a través del archivo `mqclient.ini`.

Nota: Si las propiedades de reconexión se establecen en el objeto `ConnectionFactory` y, también, en la CCDT, la regla de prioridad es la siguiente. Si el valor predeterminado de la propiedad de lista de nombres de conexión está establecido en el objeto `ConnectionFactory`, la CCDT tiene prioridad. Si la lista de nombres de conexión no está establecida en su valor predeterminado, los valores de propiedad establecidos en el objeto `ConnectionFactory` tienen prioridad. El valor predeterminado de la lista de nombres de conexión es `localhost(1414)`.

Escritura de aplicaciones de XMS .NET

En los temas de esta sección se proporciona información para ayudarle a escribir aplicaciones de XMS .NET.

Acerca de esta tarea

En esta sección se proporciona información que es específica para la escritura de aplicaciones XMS .NET. Si desea información general sobre cómo escribir aplicaciones XMS, consulte “Cómo escribir aplicaciones de XMS” en la página 650.

En esta sección se incluyen los temas siguientes:

- “Tipos de datos para .NET” en la página 670
- “Operaciones gestionadas o no gestionadas en .NET” en la página 671
- “Destinos en .NET” en la página 671
- “Propiedades en .NET” en la página 672
- “Manejo de propiedades no existentes en .NET” en la página 673
- “Manejo de errores en .NET” en la página 673
- “Utilización de escuchas de mensajes y excepciones en .NET” en la página 673

Tipos de datos para .NET

XMS .NET admite System.Boolean, System.Byte, System.SByte, System.Char, System.String, System.Single, System.Double, System.Decimal, System.Int16, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64 y System.Object. Los tipos de datos para XMS .NET son diferentes de los tipos de datos para XMS C/C++. Puede utilizar este tema para identificar los tipos de datos correspondientes.

La tabla siguiente muestra los tipos de datos C/C++ de XMS .NET y XMS correspondientes y los describe brevemente.

Tabla 86. Tipos de datos para C/C++ de XMS .NET y XMS

Tipo XMS .NET	Tipo C/C++ XMS	Descripción
System.SByte	xmsSBYTE xmsINT8	Valor de 8-bits firmado
System.Byte	xmsBYTE xmsUINT8	Valor de 8-bits sin firmar
System.Int16	xmsINT16 xmsSHORT	Valor de 16-bits firmado
System.UInt16	xmsUINT16 xmsUSHORT	Valor de 16-bits sin firmar
System.Int32	xmsINT32 xmsINT	Valor de 32-bits firmado
System.UInt32	xmsUINT32 xmsUINT	Valor de 32-bits sin firmar
System.Int64	xmsLONG xmsINT64	Valor de 64-bits firmado
System.UInt64	xmsULONG xmsUINT64	Valor de 64-bits sin firmar

Tabla 86. Tipos de datos para C/C++ de XMS .NET y XMS (continuación)		
Tipo XMS .NET	**Tipo C/C++ XMS**	**Descripción**
System.Char	xmsCHAR16	Carácter de 16-bits sin firmar (Unicode para .NET)
System.Single	xmsFLOAT	Flotante IEEE de 32-bits
System.Double	xmsDOUBLE	Flotante IEEE de 64-bits
System.Boolean	xmsBOOL	Un valor True/False
No aplicable	xmsCHAR	Valor de 8-bits firmado o sin firmar (la firma depende de la plataforma)
System.Decimal	No aplicable	Entero firmado de 96-bits por 10^0 hasta 10^{28}
System.Object	No aplicable	Base de todos los tipos
System.String	No aplicable	Tipo de serie

Operaciones gestionadas o no gestionadas en .NET

El código gestionado se ejecuta de forma exclusiva dentro del entorno de tiempo de ejecución de lenguaje común de .NET y depende por completo de los servicios proporcionados por ese tiempo de ejecución. Una aplicación se clasifica como no gestionada si alguna parte de la aplicación se ejecuta o llama a servicios fuera del entorno de tiempo de ejecución de lenguaje común de .NET.

Actualmente, determinas funciones avanzadas no se pueden soportar dentro del entorno gestionado de .NET.

Si la aplicación requiere algunas funciones que, actualmente, no están soportadas en el entorno totalmente gestionado, puede cambiar la aplicación para utilizar el entorno no gestionado sin necesitar un cambio sustancial en la aplicación. Sin embargo, debería tener en cuenta que la pila XMS utiliza el código no gestionado cuando se realiza esta selección.

Conexiones a un gestor de colas IBM MQ

Las conexiones gestionadas a WMQ_CM_CLIENT no darán soporte a comunicaciones no TCP y la compresión de canal. Sin embargo, se podría dar soporte a estas conexiones mediante el uso de una conexión no gestionada (WMQ_CM_CLIENT_UNMANAGED). Para obtener más información, consulte “Desarrollo de aplicaciones .NET” en la página 571.

Si crea una fábrica de conexiones a partir de un objeto administrado en un entorno no gestionado, debe cambiar manualmente el valor para la modalidad de conexión a XMSC_WMQ_CM_CLIENT_UNMANAGED.

Conexiones a un motor de mensajería del bus de integración de servicios WebSphere Application Server

Las conexiones a un motor de mensajería del bus de integración de servicios WebSphere Application Server que requieren el uso del protocolo SSL (incluido HTTPS) actualmente no están soportadas como código gestionado.

Destinos en .NET

En .NET, los destinos se crean de acuerdo con el tipo de protocolo y solo se pueden utilizar en el tipo de protocolo para el cual se han creado.

Se proporcionan dos funciones para crear destinos, una para temas y una para colas:

- `IDestination CreateTopic(String topic);`

- `IDestination CreateQueue(String queue);`

Estas funciones están disponibles en los dos objetos siguientes en la API:

- ISession
- XMSFactoryFactory

En ambos casos, estos métodos pueden aceptar una serie de estilo de URI, que puede incluir parámetros, en el formato siguiente:

```
"topic://some/topic/name?priority=5"
```

De forma alternativa, estos métodos pueden aceptar solo un nombre de destino, es decir, un nombre sin un prefijo topic:// o queue:// y sin parámetros.

Esto significa que la serie de estilo de URI siguiente:

```
CreateTopic("topic://some/topic/name");
```

generaría el mismo resultado que el nombre de destino siguiente:

```
CreateTopic("some/topic/name");
```

Por lo que respecta al bus de integración de servicios WebSphere Application Server JMS, los temas también se pueden especificar de forma abreviada, que incluye tanto*topicname* como *topicspace*, pero no puede incluir parámetros:

```
CreateTopic("topicspace:topicname");
```

Propiedades en .NET

Una aplicación .NET utiliza los métodos de la interfaz PropertyContext para obtener y establecer las propiedades de objetos.

La interfaz PropertyContext encapsula métodos que obtienen y establecen propiedades. Estos métodos sonheredados, de forma directa o indirecta, por las clases siguientes:

- BytesMessage
- Conexión
- ConnectionFactory
- ConnectionMetaData
- Destino
- MapMessage
- Mensaje
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Sesión (Session).
- StreamMessage
- TextMessage

Si una aplicación establece el valor de una propiedad, el nuevo valor sustituye cualquier valor previo que tuviera la propiedad.

Para obtener más información sobre las propiedades de XMS, consulte Propiedades de objetos XMS.

Para su facilidad de uso, los nombres y valores de propiedad de XMS en XMS están predefinidos como constantes públicos en una estructura llamada XMSC. Los nombres de estas constantes tienen el formato XMSC.*constante*; por ejemplo, XMSC.USERID (una constante de nombre de propiedad) y XMSC.DELIVERY_AS_APP (un valor constante).

Además, puede acceder a constantes de IBM MQ utilizando la estructura IBM.XMS.MQC. Si el espacio de nombre IBM.XMS ya se ha importado, puede acceder a los valores para estas propiedades con el formato MQC.*constante*. Por ejemplo, MQC.MQRO_COA_WITH_FULL_DATA.

Además, si tiene una aplicación híbrida que utiliza ambas clases, XMS .NET y IBM MQ, para .NET y que importa ambos espacios de nombres, IBM.XMS e IBM.WMQ, debe cualificar por completo el espacio de nombres de estructura MQC para garantizar que cada aparición es única.

Actualmente, algunas funciones no está soportadas en el entorno .NET gestionado. Consulte “Operaciones gestionadas o no gestionadas en .NET” en la página 671 si desea más detalles.

Manejo de propiedades no existentes en .NET

El manejo de propiedades no existentes en XMS .NET es ampliamente compatible con la especificación JMS y, también, con las implementaciones C y C++ de XMS.

En JMS, el acceso a una propiedad no existente puede generar una excepción del sistema Java cuando un método intenta convertir el valor no existente (nulo) al tipo necesario. Si una propiedad no existe, se producen las excepciones siguientes:

- getStringProperty y getObjectProperty devuelven null
- getBooleanProperty devuelve false porque Boolean.valueOf(null) devuelve false
- getIntProperty etc.throw java.lang.NumberFormatException porque Integer.valueOf(null) lanza la excepción

Si una propiedad no existe en XMS .NET, se producen las excepciones siguientes:

- GetStringProperty y GetObjectProperty (y GetBytesProperty) devuelven null (que es el mismo que Java)
- GetBooleanProperty lanza System.NullReferenceException
- GetIntProperty etc. lanza System.NullReferenceException

Esta implementación es diferente de Java, pero es ampliamente compatible con la especificación JMS y con las interfaces C y C++ de XMS. Al igual que la implementación de Java, XMS .NET propaga cualquier excepción desde la llamada System.Convert al interlocutor. Sin embargo, a diferencia de Java, XMS lanza de forma explícita NullReferenceExceptions, en lugar de solo utilizar el comportamiento nativo de la infraestructura de .NET pasando un nulo a las rutinas de conversión del sistema.
Si la aplicación establece una propiedad en una serie como "abc" y llama a GetIntProperty, la excepción System.FormatException lanzada por Convert.ToInt32("abc") se propaga al interlocutor, que es compatible con Java. MessageFormatException solo se lanza si los tipos utilizados para setProperty y getProperty son incompatibles. Este comportamiento también es compatible con Java.

Manejo de errores en .NET

Todas las excepciones de XMS .NET se han derivado de System.Exception. Las llamadas del método XMS pueden lanzar excepciones específicas de XMS como, por ejemplo, MessageFormatException, excepciones generales XMSExceptions o excepciones del sistema como NullReferenceException.

Escriba aplicaciones para capturar cualquiera de estos errores, ya sea en bloques catch específicos o en bloques catch `System.Exception` generales, según sea lo más apropiado para los requisitos de las aplicaciones.

Utilización de escuchas de mensajes y excepciones en .NET

Una aplicación .NET utiliza un escucha de mensajes para recibir mensajes de forma asíncrona, y utiliza un escucha de excepción para que se le notifique un problema con una conexión de forma asíncrona.

Acerca de esta tarea

La funcionalidad de los escuchas de mensajes y de excepciones es la misma para .NET y para C++. Sin embargo, existen algunas pequeñas diferencias de implementación.

Procedimiento

- Para configurar un escucha de mensajes para poder recibir mensajes de forma asíncrona, complete los pasos siguientes:
  a) Definir un método que coincida con la firma delegada del escucha de mensajes.
     El método que defina puede ser estático o un método de instancia y se puede definir en cualquier clase accesible. La firma delegada es la siguiente:

```
public delegate void MessageListener(IMessage msg);
```

     y, por lo tanto, podría definir el método como:

```
void SomeMethodName(IMessage msg);
```

  b) Instancie este método como delegado utilizando algo similar al ejemplo siguiente:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

  c) Registrar el delegado con uno o más consumidores estableciéndolo en la propiedad MessageListener del consumidor

```
consumer.MessageListener = OnMsgMethod;
```

     Puede eliminar el delegado volviendo a establecer MessageListener en null:

```
consumer.MessageListener = null;
```

- Para configurar un escucha de excepciones, complete los pasos siguientes.
  El escucha de excepción funciona de una forma muy similar que el escucha de mensajes, pero tiene una definición de delegado diferente y se asigna a la conexión, en lugar de al consumidor de mensajes. Esto es igual que para C++.
  a) Defina el método.
     La firma delegada es la siguiente:

```
public delegate void ExceptionListener(Exception ex);
```

     y, por lo tanto, el método definido podría ser:

```
void SomeMethodName(Exception ex);
```

  b) Instancie este método como delegado utilizando algo similar al ejemplo siguiente:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) Registre el delegado con la conexión estableciendo su propiedad ExceptionListener:

```
connection.ExceptionListener = OnExMethod ;
```

     Puede eliminar el delegado restableciendo ExceptionListener en:

```
null:  connection.ExceptionListener = null;
```

Cómo trabajar con objetos administrados de XMS .NET

En los temas de esta sección se proporciona información sobre objetos administrados. Las aplicaciones XMS pueden recuperar definiciones de objeto de un repositorio central de objetos administrados y utilizarlas para crear fábricas de conexiones y destinos.

Acerca de esta tarea

En esta sección se proporciona información para ayudar a crear y gestionar objetos administrados, que describen los tipos del repositorio de objetos administrados que admite XMS. En la sección también se explica cómo una aplicación XMS realiza una conexión a un repositorio de objetos administrados para recuperar los objetos administrados necesarios.

En esta sección se incluyen los temas siguientes:

- "XMS .NET Tipos soportados de repositorio de objetos administrados" en la página 675
- "XMS .NET Correlación de propiedades para objetos administrados" en la página 676
- "XMS .NET Propiedades necesarias para objetos ConnectionFactory administrados" en la página 678
- "XMS .NET Propiedades necesarias para objetos Destination administrados" en la página 679
- "XMS .NET Creación de objetos administrados" en la página 679
- "XMS .NET Creación de objetos InitialContext" en la página 680
- "XMS .NET Propiedades InitialContext" en la página 680
- "Formato URI para contextos iniciales de XMS" en la página 681
- "Servicio web de búsqueda JNDI para XMS .NET" en la página 682
- "XMS .NET Recuperación de objetos administrados" en la página 682

XMS .NET Tipos soportados de repositorio de objetos administrados

Los objetos administrados del sistema de archivo y LDAP se pueden utilizar para conectarse a IBM MQ y WebSphere Application Server, mientras que la Denominación COS solo se puede utilizar para conectarse al WebSphere Application Server.

Los directorios de objetos del sistema de archivo adoptan la forma de objetos serializados de Java Naming Directory Interface (JNDI). Los directorios de objetos LDAP son directorios que contienen objetos JNDI. Los directorios de objetos del sistema de archivos y LDAP pueden ser administrados por IBM MQ Explorer, que se proporciona con IBM MQ y posteriores. Tanto el sistema de archivos como los directorios de objetos LDAP se pueden utilizar para administrar conexiones de cliente centralizando fábricas de conexiones y destinos de IBM MQ . El administrador de red puede desplegar varias aplicaciones que hacen referencia al mismo repositorio central y que se actualizan de forma automática para reflejar los cambios en los valores de conexión realizados en el repositorio central.

Un directorio de denominación COS contiene fábricas de conexiones y destinos WebSphere Application Server service integration bus y se puede administrar utilizando la consola de administración de WebSphere Application Server. Para que una aplicación XMS recupere objetos del directorio de denominación COS, se debe desplegar un servicio web de búsqueda JNDI. Este servicio web no está disponible en todos los WebSphere Application Server service integration technologies. Consulte la documentación del producto para obtener detalles.

Nota: Reinicie las conexiones de aplicaciones para que entren en vigor los cambios en el directorio de objeto.

XMS .NET Correlación de propiedades para objetos administrados

Para permitir que las aplicaciones XMS .NET utilicen las definiciones de objeto de destino y fábrica de conexiones de IBM MQ JMS y WebSphere Application Server , las propiedades que se recuperan de estas definiciones deben correlacionarse con las propiedades XMS correspondientes que se pueden establecer en las fábricas de conexiones y destinos de XMS .

Para crear, por ejemplo, una fábrica de conexiones XMS con propiedades que se recuperan de una fábrica de conexiones JMS de IBM MQ , las propiedades deben estar correlacionadas entre las dos.

Todas las correlaciones de propiedad se realizan automáticamente.

Tabla 87 en la página 676 muestra las correlaciones entre algunas de las propiedades más comunes de fábricas de conexiones y destinos. Las propiedades que se muestran en esta tabla son sólo un pequeño conjunto de ejemplos, y no todas las propiedades que se muestran son relevantes para todos los tipos de conexión y servidores.

Tabla 87. Ejemplos de correlación de nombres para propiedades de fábrica de conexiones y destino

Nombre de propiedad IBM MQ JMS	**Nombre de propiedad XMS**	**Nombre de propiedad WebSphere Application Server service integration bus**
PERSISTENCE (PER)	XMSC_DELIVERY_MODE	
EXPIRY (EXP)	XMSC_TIME_TO_LIVE	
PRIORITY (PRI)	XMSC_PRIORITY	
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

Nota: Las propiedades que se muestran en la Tabla 88 en la página 676 son aplicables para JMS, así como para XMS .NET.

Tabla 88. Propiedades de XMS .NET

Propiedad	**Tipo de objeto**				
	CF	**QCF**	**TCF**	**Cola**	**Tema**
APPLICATIONNAME	Y	Y	Y	No disponible	No disponible
ASYNCEXCEPTION	Y	Y	Y	No disponible	No disponible
CCDTURL	Y	Y	Y	No disponible	No disponible
Canal	Y	Y	Y	No disponible	No disponible
ConnectionNameList	Y	Y	Y	No disponible	No disponible
CLIENTRECONNECTOPTIONS	Y	Y	Y	No disponible	No disponible
CLIENTRECONNECTTIMEOUT	Y	Y	Y	No disponible	No disponible
clientId	No disponible	Y	No disponible	No disponible	No disponible
COMPHDR "1" en la página 677	Y	No disponible	Y	No disponible	No disponible

Tabla 88. Propiedades de XMS .NET (continuación)					
Propiedad	**Tipo de objeto**				
	CF	**QCF**	**TCF**	**Cola**	**Tema**
MENSAJE "1" en la página 677	Y	Y	Y	No disponible	No disponible
CONNOPT "1" en la página 677	Y	Y	Y	No disponible	No disponible
CONNTAG "1" en la página 677	Y	Y	Y	No disponible	No disponible
DESCRIPCIÓN "1" en la página 677	No disponible	Y	No disponible	Y	Y
CADUCIDAD "1" en la página 677	No disponible	No disponible	No disponible	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
Nombre de host	No disponible	Y	No disponible	No disponible	No disponible
LOCALADDRESS	No disponible	Y	No disponible	No disponible	No disponible
Persistencia	No disponible	No disponible	No disponible	Y	Y
Puerto	No disponible	Y	No disponible	No disponible	No disponible
Prioridad "1" en la página 677	No disponible	No disponible	No disponible	Y	Y
PROVIDERVERSION "1" en la página 677	No disponible	Y	No disponible	No disponible	No disponible
QMANAGER	Y	Y	Y	Y	No disponible
Cola "1" en la página 677	No disponible	No disponible	No disponible	Y	No disponible
SHARECONVALLOWED	Y	Y	Y	No disponible	No disponible
tema "1" en la página 677	No disponible	No disponible	No disponible	No disponible	Y
Transport "1" en la página 677	No disponible	Y	No disponible	No disponible	No disponible

Nota:

1. Estas propiedades no tienen propiedades de nivel de aplicación, pero se pueden establecer opcionalmente utilizando propiedades administradas.

OutboundSNI propiedad

A partir de IBM MQ 9.3.0, puede establecer la propiedad XMSC_WMQ_OUTBOUND_SNI, que establece la propiedad **OutboundSNI** en una aplicación.

La propiedad XMSC_WMQ_OUTBOUND_SNI_PROPERTY toma los valores siguientes:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL, que se correlaciona con "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME, que se correlaciona con "HOSTNAME"
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK, que se correlaciona con "*"

Además, puede establecer la propiedad **OutboundSNI** utilizando la variable de entorno MQOUTBOUND_SNI, que toma los valores siguientes:

- CHANNEL
- HOSTNAME
- *

Nota: El valor predeterminado de la propiedad es XMSC_WMQ_OUTBOUND_SNI_CHANNEL si no se establece ningún valor específico.

El orden de prioridad para establecer la propiedad **OutboundSNI** en el nodo gestionado es:

1. Propiedad de nivel de aplicación
2. Variable de entorno

Para la propiedad **OutboundSNI** en un nodo no gestionado, sólo se da soporte a `mqclient.ini` .

XMS .NET Propiedades necesarias para objetos ConnectionFactory administrados

Cuando una aplicación crea una fábrica de conexiones, se debe definir un número de propiedades para crear una conexión a un servidor de mensajería.

Las propiedades listadas en las tablas siguientes son el mínimo necesario para establecer para una aplicación para crear una conexión a un servidor de mensajería. Si desea personalizar la forma en la que se crea una conexión, la aplicación puede establecer cualquier propiedad adicional del objeto ConnectionFactory, según sea necesario. Para obtener más información, consulte Propiedades de ConnectionFactory. Se incluye una lista completa de propiedades disponibles.

Conexión a un gestor de colas de IBM MQ

Tabla 89. Valores de propiedad para objetos ConnectionFactory administrados para conexiones con un gestor de colas IBM MQ

Propiedad XMS necesaria	**Propiedad de IBM MQ JMS equivalente necesaria**
XMSC_CONNECTION_TYPE	XMS resuelve el nombre de clase de la fábrica de conexiones y la propiedad TRANSPORT (TRAN).
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	Nombre del gestor de colas

Conexión en tiempo real a un intermediario

Tabla 90. Valores de propiedad para objetos ConnectionFactory administrados para conexiones en tiempo real a un intermediario

XMS necesario	**Propiedad de IBM MQ JMS equivalente necesaria**
XMSC_CONNECTION_TYPE	XMS resuelve el nombre de clase de la fábrica de conexiones y la propiedad TRANSPORT (TRAN).
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

Conexión a un WebSphere Application Server service integration bus

Tabla 91. Valores de propiedad para objetos ConnectionFactory administrados para conexiones a un WebSphere Application Server service integration bus	
Propiedad XMS	**Descripción**
XMSC_CONNECTION_TYPE	El tipo del servidor de mensajería al que se conecta una aplicación.. Esto se determina a partir del nombre de clase de fábrica de conexiones.
XMSC_WPM_BUS_NAME	Para una fábrica de conexiones, el nombre del bus de integración de servicios al que se conecta la aplicación o, para un destino, el nombre del bus de integración de servicios en el cual existe el destino.

XMS .NET Propiedades necesarias para objetos Destination administrados

Una aplicación que está creando un destino debe establecer varias propiedades para la aplicación en un objeto Destination administrado.

Tabla 92. Valores de propiedad para objetos Destination administrados		
Tipo de conexión	**Propiedad**	**Descripción**
Gestor de colas IBM MQ	QUEUE (QU) TOPIC (TOP)	La cola a la que desea conectarse El tema que utiliza la aplicación como destino
Conexión en tiempo real a un intermediario	TOPIC (TOP)	El tema que utiliza la aplicación como destino
WebSphere Application Server service integration bus	topicName queueName	Si la aplicación está conectándose a un tema Si la aplicación se está conectando a una cola

XMS .NET Creación de objetos administrados

Las definiciones de los objetos ConnectionFactory y Destination que necesitan las aplicaciones XMS para establecer una conexión a un servidor de mensajería se deben crear utilizando las herramientas administrativas apropiadas.

Antes de empezar

Si desea más detalles sobre los distintos tipos del repositorio de objetos administrados que admite XMS, consulte “XMS .NET Tipos soportados de repositorio de objetos administrados” en la página 675.

Acerca de esta tarea

Para crear los objetos administrados para IBM MQ utilice IBM MQ Explorer o la herramienta de administración JMS IBM MQ (JMSAdmin).

Para crear los objetos administrados para IBM MQ o IBM Integration Bus, utilice la herramienta de administración JMS IBM MQ (JMSAdmin).

Para crear objetos administrados para WebSphere Application Server service integration bus, utilice la consola de administración de WebSphere Application Server.

En las herramientas administrativas, la propiedad se conoce como **APPLICATIONNAME**, o **APPNAME** para abreviar.

Nota: No puede utilizar JMSAdmin para establecer TRANSPORT(UNMANAGED). Por lo tanto, para obtener un cliente XMS no gestionado utilizando un nombre de aplicación elegido administrativamente, debe especificar el mandato siguiente:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

Los pasos siguientes resumen lo qué se debe hacer para crear objetos administrados.

Procedimiento

1. Crear una fábrica de conexiones y definir las propiedades necesarias para crear una conexión de la aplicación al servidor elegido.

   Las propiedades mínimas que requiere XMS para realizar una conexión están definidas en “XMS .NET Propiedades necesarias para objetos ConnectionFactory administrados” en la página 678.
2. Crear el destino necesario en el servidor de mensajería, al que se conecta la aplicación:
   - Para una conexión con un gestor de colas de IBM MQ , cree una cola o un tema.
   - Para una conexión en tiempo real a un intermediario, cree un tema.
   - Para una conexión a un WebSphere Application Server service integration bus, cree una cola o un tema.

   Las propiedades mínimas que requiere XMS para realizar una conexión están definidas en “XMS .NET Propiedades necesarias para objetos Destination administrados” en la página 679.

XMS .NET Creación de objetos InitialContext

Una aplicación debe crear un contexto inicial que se va a utilizar para realizar una conexión con el repositorio de objetos administrados para recuperar los objetos administrados necesarios.

Acerca de esta tarea

Un objeto InitialContext encapsula una conexión con el repositorio. La API XMS proporciona métodos para realizar las tareas siguientes:

- Crear un objeto InitialContext
- Busca un objeto administrado en el repositorio de objetos administrados.

Procedimiento

- Para obtener detalles adicionales sobre cómo crear un objeto InitialContext, consulte InitialContext para .NET y Propiedades de InitialContext.

XMS .NET Propiedades InitialContext

Los parámetros del constructor InitialContext incluyen la ubicación del repositorio de objetos administrados, dada como un identificador uniforme de recursos (URI). Para que una aplicación pueda establecer una conexión con el repositorio, puede ser necesario proporcionar más información que la información incluida en el URI.

En JNDI y en la implementación de .NET de XMS, la información adicional se proporciona en un entorno de tabla hash al constructor.

La ubicación del repositorio de objetos administrados se define en la propiedad XMSC_IC_URL. Esta propiedad normalmente se pasa en la llamada Create, pero se puede modificar para conectarse a un directorio de denominación diferente antes de la búsqueda. Para contextos de FileSystem o LDAP, esta propiedad define la dirección del directorio. Para la denominación COS, esta es la dirección del servicio web que utiliza estas propiedades para conectarse al directorio JNDI.

Las propiedades siguientes se pasan sin modificar al servicio web que las utilizará para conectarse al directorio JNDI.

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

Formato URI para contextos iniciales de XMS

La ubicación del repositorio de objetos administrados se proporciona como un identificador uniforme de recursos (URI). El formato del URI depende del tipo de contexto.

Contexto FileSystem

Para el contexto FileSystem, el URL proporciona la ubicación del directorio basado en el sistema de archivos. La estructura del URL se define mediante RFC 1738, *Localizadores uniformes de recursos (URL)*: el URL tiene el prefijo `file://` y la sintaxis que aparece detrás de este prefijo es una definición válida de un archivo que se puede abrir en el sistema en el cual se está ejecutando XMS.

Esta sintaxis puede ser específica de la plataforma y puede utilizar separadores '/ o separadores '\'. Si utiliza '\', se debe añadir a cada separador un carácter de escape utilizando un '\' adicional. Esto impide a la infraestructura de .NET intentar interpretar el separador como un carácter de escape para lo que aparece después.

Estos ejemplos ilustran esta sintaxis:

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

Contexto LDAP

Para el contexto LDAP, la estructura básica del URL se define mediante RFC 2255, *El formato del URL deLDAP*, que el prefijo `ldap://` que no distingue entre mayúsculas y minúsculas

La sintaxis precisa se ilustra en el ejemplo siguiente:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Esta sintaxis es la que se define en RFC, pero sin el soporte para ningún atributo, ámbito, filtro o extensión.

Entre los ejemplos de esta sintaxis se incluye:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

Contexto WSS

Para el contexto WSS, el URL tiene el formato de un punto final de servicios web, con el prefijo `http://`.

De forma alternativa, puede utilizar el prefijo `cosnaming://` o `wsvc://`,

Estos dos prefijos se interpretan como que utiliza un contexto WSS con el URL al que se accede a través de http, lo que permite que se derive el tipo de contexto inicial fácil y directamente desde el URL.

Entre los ejemplos de esta sintaxis se incluye lo siguiente:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Servicio web de búsqueda JNDI para XMS .NET

Para acceder a un directorio de denominación COS desde XMS, se debe desplegar un servicio web de búsqueda JNDI en un servidor WebSphere Application Server service integration bus. Este servicio web convierte la información de Java del servicio de denominación COS en un formulario que la aplicación XMS puede leer.

El servicio web se proporciona en el archivo de archivador empresarial SIBXJndiLookupEAR.ear, situado dentro del directorio de instalación. Para el release actual de IBM MQ Message Service Client (XMS) for .NET, SIBXJndiLookupEAR.ear se puede encontrar en el directorio *install_dir*\java\lib . Esto se puede instalar en un servidor WebSphere Application Server service integration bus utilizando la consola de administración o la herramienta de scripts wsaadmin. Consulte la documentación del producto si desea más información sobre cómo desplegar aplicaciones de servicio web.

Para definir el servicio web en aplicaciones XMS, simplemente tendrá que establecer la propiedad XMSC_IC_URL del objeto InitialContext en el URL de punto final de servicio web. Por ejemplo, si el servicio web se ha desplegado en un host de servidor llamado MyServer, un ejemplo de un URL de punto final de servicio web:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

La definición de la propiedad XMSC_IC_URL permite a las llamadas de búsqueda InitialContext invocar al servicio web en el punto final definido que, a su vez, busca el objeto administrado necesario desde un servicio de denominación COS.

Las aplicaciones .NET pueden utilizar el servicio web. El despliegue del lado del servidor es el mismo para XMS C, /C++ y XMS .NET.XMS .NET invoca el servicio web directamente a través de Microsoft .NET Framework.

XMS .NET Recuperación de objetos administrados

XMS recupera un objeto administrado del repositorio utilizando la dirección proporcionada cuando se crea el objeto InitialContext, o en las propiedades InitialContext.

Los objetos que se van a recuperar pueden tener los tipos de nombres siguientes:

- Un nombre sencillo que describe el objeto Destination, por ejemplo, un destino de cola llamado SalesOrders
- Un nombre compuesto, que puede estar formado de SubContexts, separados por '/', y debe acabar con el nombre de objeto. Un ejemplo de un nombre compuesto es "Warehouse/PickLists/DispatchQueue2" donde Warehouse y Picklists son SubContexts en el directorio de denominación, y DispatchQueue2 es el nombre de un objeto Destination.

Cómo evitar que las aplicaciones utilicen una versión más nueva de XMS

De forma predeterminada, cuando está instalada una versión más reciente de XMS, las aplicaciones que utilizan la versión anterior cambian a la versión más reciente sin tener que volver a compilar. Sin embargo, puede evitar que las aplicaciones utilicen la versión más nueva estableciendo un atributo en el archivo de configuración de la aplicación.

Acerca de esta tarea

La característica de coexistencia de varias versiones garantiza que la instalación de una versión de XMS más nueva no sobrescribe la versión anterior de XMS. En lugar de esto, coexisten varias instancias de

conjuntos similares de XMS .NET en la Global Assembly Cache (GAC), pero tienen números de versión diferentes. De forma interna, la GAC utiliza un archivo de política para direccionar las llamadas de aplicación a la última versión de XMS. Las aplicaciones se ejecutan sin necesidad de recompilar y pueden utilizar nuevas características disponibles en la versión más nueva de XMS .NET.

Procedimiento

- Si es necesaria una aplicación para utilizar la versión más antigua de XMS .NET, establezca el atributo `publisherpolicy` en no en el archivo de configuración de la aplicación.

  Nota: Un archivo de configuración de aplicación es un archivo con un nombre que está formado por el nombre del programa ejecutable con el que está relacionado el archivo, con el sufijo .config. Por ejemplo, el archivo de configuración de aplicación para text.exe tendría el nombre text.exe.config.

  Sin embargo, en cualquier momento, todas las aplicaciones de un sistema utilizan la misma versión de XMS .NET.

Protección de comunicaciones para aplicaciones XMS

En esta sección se proporciona información sobre cómo configurar comunicaciones seguras para permitir a las aplicaciones XMS conectarse a través de Secure Sockets Layer (SSL) a un motor de mensajería WebSphere Application Server service integration bus o un gestor de colas IBM MQ.

Acerca de esta tarea

La sección contiene los temas siguientes:

- “Conexiones seguras a un gestor de colas IBM MQ” en la página 683
- “Correlaciones de nombres CipherSuite y CipherSpec para conexiones de XMS con un gestor de colas IBM MQ” en la página 684
- “Conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus” en la página 684
- “Correlaciones de nombres CipherSuite y CipherSpec para conexiones a un WebSphere Application Server service integration bus” en la página 685

Conexiones seguras a un gestor de colas IBM MQ

Para permitir que una aplicación XMS .NET realice conexiones seguras a un gestor de colas IBM MQ, las propiedades relevantes deben estar definidas en el objeto ConnectionFactory.

El protocolo utilizado en la negociación de cifrado puede ser Secure Sockets Layer (SSL) o Transport Layer Security (TLS), en función de la CipherSuite que especifique en el objeto ConnectionFactory.

En la tabla siguiente se muestran las propiedades ConnectionFactory para conexiones que utilizan SSL con un gestor de colas IBM MQ , con una breve descripción:

Tabla 93. Propiedades de ConnectionFactory para conexiones a un gestor de colas IBM MQ a través de SSL

Nombre de la propiedad	Descripción
XMSC_WMQ_SSL_CERT_STORES	Las ubicaciones de los servidores que contienen las listas de revocaciones de certificados (CRL) que se van a utilizar en una conexión SSL a un gestor de colas.
XMSC_WMQ_SSL_CIPHER_SPEC	El nombre de la CipherSpec que se va a utilizar en una conexión segura a un gestor de colas.
XMSC_WMQ_SSL_CIPHER_SUITE	El nombre de la CipherSuite que se va a utilizar en una conexión TLS a un gestor de colas. El protocolo utilizado en la negociación de la conexión segura depende de la CipherSuite especificada.

Tabla 93. Propiedades de ConnectionFactory para conexiones a un gestor de colas IBM MQ a través de SSL (continuación)	
Nombre de la propiedad	**Descripción**
XMSC_WMQ_SSL_CRYPTO_HW	Detalles de configuración para el hardware de cifrado conectado al sistema cliente.
XMSC_WMQ_SSL_FIPS_REQUIRED	El valor de esta propiedad determina si una aplicación puede o no puede utilizar suites de cifrado no compatibles con FIPS. Si esta propiedad se establece en true (verdadero), solo se utilizan algoritmos FIPS para la conexión cliente/servidor.
XMSC_WMQ_SSL_KEY_REPOSITORY	La ubicación de un archivo de base de datos de claves en el cual se almacenan claves y certificados.
XMSC_WMQ_SSL_KEY_RESETCOUNT	El KeyResetCount representa el número total de bytes sin cifrado enviados y recibidos en una conversación SSL antes de renegociar la clave secreta.
XMSC_WMQ_SSL_PEER_NAME	El nombre de igual que se va a utilizar en una conexión SSL a un gestor de colas.

Correlaciones de nombres CipherSuite y CipherSpec para conexiones de XMS con un gestor de colas IBM MQ

La propiedad InitialContext convierte entre la propiedad de fábrica de conexiones JMSAdmin SSLCIPHERSUITE y la propiedad XMSC_WMQ_SSL_CIPHER_SPEC de XMS casi equivalente. Es necesaria una conversión similar si especifica un valor para XMSC_WMQ_SSL_CIPHER_SUITE pero omite el valor paraXMSC_WMQ_SSL_CIPHER_SPEC.

Tabla 94 en la página 684 lista todas las CipherSpecs disponibles y sus equivalentes de JSSE CipherSuite.

Tabla 94. CipherSpecs disponibles y sus equivalentes de JSSE CipherSuite	
CipherSpec	**JSSE CipherSuite equivalente**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.

Conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus

Para permitir que una aplicación XMS .NET realice conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus, en el objeto ConnectionFactory deben estar definidas las propiedades relevantes.

XMS proporciona soporte SSL y HTTPS para conexiones a un WebSphere Application Server service integration bus. SSL y HTTPS proporcionan conexiones seguras para la autenticación y la confidencialidad.

Al igual que la seguridad WebSphere, la seguridad XMS se configura con respecto a los estándares de seguridad y convenios de denominación JSSE, que incluyen el uso de CipherSuites para especificar los algoritmos que se utilizan cuando se negocia una conexión segura. El protocolo utilizado en la negociación de cifrado puede ser SSL o TLS, en función de la CipherSuite que especifique en el objeto ConnectionFactory.

Tabla 95 en la página 685 lista las propiedades que se deben definir en el objeto ConnectionFactory.

Tabla 95. Propiedades de ConnectionFactory para conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus

Nombre de la propiedad	Descripción
XMSC_WPM_SSL_CIPHER_SUITE	El nombre de la CipherSuite que se va a utilizar en una conexión TLS con un motor de mensajería de WebSphere Application Server service integration bus . El protocolo utilizado en la negociación de la conexión segura depende de la CipherSuite especificada.
XMSC_WPM_SSL_KEYRING_LABEL	El certificado que se va a utilizar al autenticarse con el servidor.

A continuación aparece un ejemplo de propiedades ConnectionFactory para conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus:

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Donde nombre_cadena se debería establecer en BootstrapTunneledSecureMessaging o BootstrapSecureMessaging, y número_puerto es el número del puerto donde el servidor de programa de arranque escucha solicitudes entrantes.

A continuación, se muestra un ejemplo de propiedades ConnectionFactory para conexiones seguras a un motor de mensajería WebSphere Application Server service integration bus con los valores de ejemplo insertados:

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

Correlaciones de nombres CipherSuite y CipherSpec para conexiones a un WebSphere Application Server service integration bus

Puesto que IBM Global Security Kit (GSKit) utiliza CipherSpecs en lugar de CipherSuites, los nombres CipherSuite de estilo JSSE especificados en la propiedad XMSC_WPM_SSL_CIPHER_SUITE deben correlacionarse con los nombres CipherSpec de estilo GSKit.

Tabla 96 en la página 685 lista la CipherSpec equivalente para cada CipherSuite reconocida.

Tabla 96. CipherSuites disponibles y sus CipherSpecs equivalentes

CipherSuite	CipherSpec equivalente
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA

Tabla 96. CipherSuites disponibles y sus CipherSpecs equivalentes (continuación)	
CipherSuite	**CipherSpec equivalente**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

Nota: Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA está en desuso. Sin embargo, se sigue pudiendo utilizar para transferir hasta 32 GB de datos antes de que se termine la conexión con el error AMQ9288. Para evitar este error, tendrá que evitar utilizar el triple DES, o habilitar el restablecimiento de clave secreta cuando se utiliza esta CipherSpec.

Mensajes de XMS

En esta sección se describen la estructura y el contenido de mensajes de XMS y se explica cómo las aplicaciones procesan mensajes de XMS.

En esta sección se incluyen los temas siguientes:

- "Partes de un mensaje XMS" en la página 686
- "Campos de cabecera en un mensaje XMS" en la página 686
- "Propiedades de un mensaje XMS" en la página 687
- "El cuerpo de un mensaje XMS" en la página 690
- "Selectores de mensaje" en la página 693
- "Correlación de mensajes XMS en mensajes IBM MQ" en la página 694

Partes de un mensaje XMS

Un mensaje XMS está formado por una cabecera, un conjunto de propiedades y un cuerpo.

Cabecera
La cabecera de un mensaje contiene campos y todos los mensajes contienen el mismo conjunto de campos de cabecera. XMS y las aplicaciones utilizan los valores de los campos de cabecera para identificar y direccionar mensajes. Si desea más información sobre campos de cabecera. consulte "Campos de cabecera en un mensaje XMS" en la página 686.

Conjunto de propiedades
Las propiedades de un mensaje especifican información adicional sobre el mensaje. Aunque todos los mensajes tienen el mismo conjunto de campos de cabecera, cada mensaje puede tener un conjunto de propiedades diferente. Para obtener más información, consulte "Propiedades de un mensaje XMS" en la página 687.

Cuerpo
El cuerpo de un mensaje contiene datos de aplicación. Para obtener más información, consulte "El cuerpo de un mensaje XMS" en la página 690.

Una aplicación puede seleccionar qué mensajes desea recibir. Mediante el uso de selectores de mensajes, que especifican los criterios de selección. Los criterios se pueden basar en los valores de determinados campos de cabecera y los valores de cualquiera de las propiedades de un mensaje. Si desea más información sobre selectores de mensajes, consulte "Selectores de mensaje" en la página 693.

Campos de cabecera en un mensaje XMS

Para permitir que una aplicación XMS intercambie mensajes con una aplicación WebSphere JMS, la cabecera de un mensaje XMS contiene los campos de cabecera de mensaje JMS.

Los nombres de estos campos de cabecera empiezan con el prefijo JMS. Para ver una descripción de los campos de cabecera de mensaje JMS, consulte la *Especificación Java Message Service*.

XMS implementa los campos de cabecera de mensaje JMS como atributos de un objeto Message. Cada campo de cabecera tiene sus propios métodos para establecer y obtener su valor. Para obtener una descripción de estos métodos, consulte IMessage. Un campo de cabecera siempre se puede leer y grabar.

Tabla 97 en la página 687 lista los campos de cabecera de mensaje JMS e indica cómo se establece el valor de cada campo para un mensaje transmitido. Algunos de los campos se establecen automáticamente mediante XMS cuando una aplicación envía un mensaje o, en el caso de JMSRedelivered, cuando una aplicación recibe un mensaje.

Tabla 97. Campos de cabecera de mensaje JMS.]

Nombre del campo de cabecera de mensaje JMS	**Cómo se establece el valor para un mensaje transmitido (con el formato de *método* [*clase*])**
JMSCorrelationID	Establecer JMSCorrelationID [Message]
JMSDeliveryMode	Enviar [MessageProducer]
JMSDestination	Enviar [MessageProducer]
JMSExpiration	Enviar [MessageProducer]
JMSMessageID	Enviar [MessageProducer]
JMSPriority	Enviar [MessageProducer]
JMSRedelivered	Recibir [MessageConsumer]
JMSReplyTo	Establecer JMSReplyTo [Message]
JMSTimestamp	Enviar [MessageProducer]
JMSType	Establecer JMSType [Message]

Propiedades de un mensaje XMS

XMS admite tres tipos de propiedad de mensaje: propiedades definidas de JMS, propiedades definidas de IBM y propiedades definidas por aplicación.

Una aplicación XMS puede intercambiar mensajes con una aplicación WebSphere JMS porque XMS admite las propiedades predefinidas siguientes de un objeto Message:

- Las mismas propiedades definidas por JMS que admite WebSphere JMS. Los nombres de estas propiedades empiezan con el prefijo JMSX.
- Las mismas propiedades definidas por IBM que admite WebSphere JMS. Los nombres de estas propiedades empiezan con el prefijo JMS_IBM_.

Cada propiedad predefinida tiene dos nombres:

- Un nombre JMS, para una propiedad definida por JMS, o un nombre WebSphere JMS, para una propiedad definida por IBM.

  Este es el nombre por el que se conoce la propiedad en JMS o WebSphere JMS y, también, es el nombre que se transmite con un mensaje que tiene esta propiedad. Una aplicación XMS utiliza este nombre para identificar la propiedad en una expresión de selector de mensajes.
- Un nombre XMS para identificar la propiedad en todas las situaciones, excepto en una expresión de selector de mensajes. Cada nombre XMS se define como una constante con nombre en la clase `IBM.XMS.XMSC`. El valor de la constante con nombre es el nombre JMS o WebSphere JMS correspondiente.

Además de las propiedades predefinidas, una aplicación XMS puede crear y utilizar su propio conjunto de propiedades de mensaje. Estas propiedades se denominan *propiedades definidas por aplicación.*

Después de que una aplicación haya creado un mensaje, las propiedades del mensaje se pueden leer y grabar. Las propiedades se siguen pudiendo leer y grabar después de que la aplicación envíe el mensaje. Cuando una aplicación recibe un mensaje, las propiedades del mensaje son de solo lectura. Si una

aplicación llama al método `Clear Properties` de la clase Message cuando las propiedades de un mensaje son de sólo lectura, las propiedades pasan a ser legibles y grabables. El método también borra las propiedades.

El mensaje recibido, cuando se envía después de borrar las propiedades del mensaje, se comportará de una forma coherente con el comportamiento del envío de un WMQ XMS for .NET BytesMessage estándar con las propiedades de mensaje borradas.

Sin embargo, esto no se recomienda porque se perderán las propiedades siguientes:

- Valor de propiedad JMS_IBM_Encoding, que implica que los datos de mensaje no se pueden descodificar con sentido.
- Valor de propiedad JMS_IBM_Format, que implica que se rompería el encadenamiento de cabecera entre la cabecera de mensaje (MQMD o la MQRFH2 nueva) y las cabeceras existentes.

Para determinar los valores de todas las propiedades de un mensaje, una aplicación puede llamar al método Obtener propiedades de la clase Message. El método crea un iterador que encapsula una lista de objetos Property, donde cada objeto Property representa una propiedad del mensaje. La aplicación puede utilizar los métodos de la clase Iterator para recuperar cada objeto Property a su vez, y puede utilizar los métodos de la clase Property para recuperar el nombre, tipo de datos y valor de cada propiedad.

Propiedades definidas por JMS de un mensaje

XMS y WebSphere JMS admiten ambos varias propiedades definidas por JMS de un mensaje.

Tabla 98 en la página 688 lista las propiedades definidas por JMS de un mensaje que admiten XMS y, también, WebSphere JMS. Si desea una descripción de las propiedades definidas por JMS, consulte *Especificación de Java Message Service*. Las propiedades definidas por JMS no son válidas para una conexión en tiempo real a un intermediario.

La tabla especifica el tipo de datos de cada propiedad e indica cómo se establece el valor de la propiedad para un mensaje transmitido. Algunas de las propiedades se establecen automáticamente mediante XMS cuando una aplicación envía un mensaje o, en el caso de JMSXDeliveryCount, cuando una aplicación recibe un mensaje.

Tabla 98. Propiedades definidas por JMS de un mensaje

Nombre XMS de la propiedad definida de JMS	**Nombre de JMS**	**Tipo de datos**	**Cómo se establece el valor para un mensaje transmitido (con el formato de *método* [*clase*])**
JMSX_APPID	JMSXAppID	System.String	Enviar [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Recibir [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Establecer propiedad de serie [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Establecer propiedad de entero [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Enviar [MessageProducer]

Propiedades definidas por IBM de un mensaje

XMS y WebSphere JMS dan soporte a varias propiedades definidas por IBM de un mensaje.

Tabla 99 en la página 689 lista las propiedades definidas de IBM de un mensaje que están soportadas por XMS y WebSphere JMS. Si desea más información sobre las propiedades definidas por IBM, consulte la documentación de producto IBM MQ o WebSphere Application Server.

La tabla especifica el tipo de datos de cada propiedad e indica cómo se establece el valor de la propiedad para un mensaje transmitido. XMS establece automáticamente algunas de las propiedades cuando una aplicación envía un mensaje.

Tabla 99. Propiedades definidas por IBM de un mensaje

Nombre XMS de la propiedad definida de IBM	**Nombre WebSphere JMS**	**Tipo de datos**	**Cómo se establece el valor para un mensaje transmitido (con el formato de *método* [*clase*])**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Recibir [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Recibir [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Recibir [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM DESTINO	JMS_IBM_ExceptionProblemDestination	System.String	Recibir [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Establecer propiedad de serie [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Establecer propiedad de entero [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Enviar [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Enviar [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Enviar [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Establecer propiedad de entero [PropertyContext]

Tabla 99. Propiedades definidas por IBM de un mensaje (continuación)			
Nombre XMS de la propiedad definida de IBM	**Nombre WebSphere JMS**	**Tipo de datos**	**Cómo se establece el valor para un mensaje transmitido (con el formato de *método* [*clase*])**
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ Nº Propuesta	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Establecer propiedad de entero [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Enviar [MessageProducer]

Propiedades de un mensaje definidas por la aplicación

Una aplicación XMS puede crear y utilizar su propio conjunto de propiedades de mensaje. Cuando una aplicación envía un mensaje, estas propiedades también se transmiten con el mensaje. Una aplicación receptora, que utiliza selectores de mensajes, puede seleccionar qué mensajes desea recibir basándose en los valores de estas propiedades.

Para permitir que una aplicación WebSphere JMS seleccione y procese los mensajes enviados por una aplicación XMS, el nombre de una propiedad definida por la aplicación debe ser compatible con las normas para formar identificadores en las expresiones de selector de mensaje. Para obtener más información, consulte “Selectores de mensajes en JMS” en la página 150. El valor de una propiedad definida por la aplicación debe tener uno de los tipos de datos siguientes: System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Double o System.String.

El cuerpo de un mensaje XMS

El cuerpo de un mensaje contiene datos de aplicación. Sin embargo, un mensaje puede no tener cuerpo, y estar formado solo de los campos de cabecera y las propiedades.

XMS admite cinco tipos de cuerpo de mensaje:

Bytes
El cuerpo contiene una corriente de bytes. Un mensaje con este tipo de cuerpo se denomina *mensaje de bytes*. La interfaz IBytesMessage contiene los métodos para procesar el cuerpo de un mensaje de bytes.

Correlación
El cuerpo contiene un conjunto de pares de nombre-valor, donde cada valor tiene un tipo de datos asociado. Un mensaje con este tipo de cuerpo se denomina *mensaje de correlación*. La interfaz IMapMessage contiene los métodos para procesar el cuerpo de un mensaje de correlación.

Objeto
El cuerpo contiene un objeto serializado Java o .NET. Un mensaje con este tipo de cuerpo se denomina *mensaje de objeto*. La interfazIObjectMessage contiene los métodos para procesar el cuerpo de un mensaje de objeto.

Corriente
El cuerpo contiene una corriente de valores, donde cada valor tiene un tipo de datos asociado. Un mensaje con este tipo de cuerpo se denomina *mensaje de corriente de datos*. La interfaz IStreamMessage contiene los métodos para procesar el cuerpo de un mensaje de corriente de datos.

Texto
El cuerpo contiene una serie. Un mensaje con este tipo de cuerpo se denomina *mensaje de texto*. La interfaz ITextMessage contiene los métodos para procesar el cuerpo de un mensaje de texto.

La interfaz IMessage es el padre de todos los objetos de mensaje y se puede utilizar en funciones de mensajería para representar cualquiera de los tipos de mensaje de XMS.

Si desea más información sobre el tamaño y los valores máximo y mínimo de cada uno de estos tipos de datos, consulte Tabla 84 en la página 666.

Mensajes de bytes

El cuerpo de un mensaje de bytes contiene una corriente de bytes. El cuerpo solo contiene los datos reales, y es responsabilidad de las aplicaciones emisoras y receptoras interpretar estos datos.

Los mensajes de bytes son útiles si una aplicación XMS debe intercambiar mensajes con aplicaciones que no están utilizando la interfaz de programación de aplicaciones XMS o JMS.

Después de que una aplicación cree un mensaje debytes, el cuerpo del mensaje es de solo escritura. La aplicación ensambla los datos de aplicación en el cuerpo llamando a los métodos de escritura apropiados de la interfaz IBytesMessage para .NET. Cada vez que la aplicación escribe un valor en la corriente de mensajes de bytes, el valor se ensambla inmediatamente después del valor anterior escrito por la aplicación. XMS mantiene un cursor interno para recordar la posición del último byte que se ensambló.

Cuando la aplicación envía el mensaje, el cuerpo del mensaje se convierte en de solo lectura. De esta forma, la aplicación puede enviar el mensaje de forma repetida.

Cuando una aplicación recibe un mensaje de bytes, el cuerpo del mensaje es de solo lectura. La aplicación puede utilizar los métodos de lectura apropiados de la interfaz IBytesMessage para leer el contenido de la corriente de mensajes de bytes. La aplicación lee los bytes de la secuencia, y XMS mantiene un cursor interno para recordar la posición del último byte que se leyó.

Si una aplicación llama al método Restablecer de la interfaz IBytesMessage cuando el cuerpo de un mensaje de bytes se puede grabar, el cuerpo pasa a ser de solo lectura. El método también vuelve a colocar el cursor al inicio de la corriente de mensajes de bytes.

Si una aplicación llama al método `Clear Body` de la interfaz IMessage para .NET cuando el cuerpo de un mensaje de bytes es de sólo lectura, el cuerpo pasa a ser grabable. El método también borra el cuerpo.

Mensajes de correlación

El cuerpo de un mensaje de correlación contiene un conjunto de pares de nombre-valor, donde cada valor tiene un tipo de datos asociado.

En cada par de nombre-valor, el nombre es una serie que identifica el valor, y el valor es un elemento de datos de aplicación que tiene uno de los tipos de datos de XMS listados en Tabla 100 en la página 693. El orden de los partes de nombre-valor no está definido. La clase MapMessage contiene los métodos para establecer y obtener pares de nombre-valor.

Una aplicación puede acceder a un par de nombre-valor de forma aleatoria especificando su nombre.

Una aplicación .NET puede utilizar la propiedad MapNames para obtener una enumeración de los nombres en el cuerpo del mensaje de correlación.

Cuando una aplicación obtiene el valor de un par de nombre-valor, el valor se puede convertir mediante XMS en otro tipo de datos. Por ejemplo, para obtener un entero del cuerpo de un mensaje de correlación, una aplicación puede llamar al método GetString de la clase MapMessage, que devuelve el entero como una serie. Las conversiones soportadas son las mismas que las que están soportadas cuando XMS convierte un valor de propiedad de un tipo a otro. Si desea más información sobre las conversiones

soportadas, consulte “Conversión implícita de un valor de propiedad de un tipo de datos a otro” en la página 666.

Después de que una aplicación crea un mensaje de correlación, el cuerpo del mensaje se puede leer y grabar. El cuerpo se sigue pudiendo leer y escribir después de que la aplicación envíe el mensaje. Cuando una aplicación recibe un mensaje de correlación, el cuerpo del mensaje es de solo lectura. Si una aplicación llama al método Borrar cuerpo de la clase Message cuando el cuerpo de un mensaje de correlación es de solo lectura, el cuerpo pasa poderse leer y grabar. El método también borra el cuerpo.

Mensajes de objeto

El cuerpo de un mensaje de objeto contiene un objeto Java serializado o un objeto .NET.

Una aplicación XMS puede recibir un mensaje de objeto, cambiar sus campos y propiedades de cabecera y, después, enviarlo a otro destino. Una aplicación también puede copiar el cuerpo de un mensaje de objeto y utilizarlo para formar otro mensaje de objeto. XMS trata el cuerpo de un mensaje de objeto como una matriz de bytes.

Después de que una aplicación crea un mensaje de objeto, el cuerpo del mensaje se puede leer y escribir. El cuerpo se sigue pudiendo leer y escribir después de que la aplicación envíe el mensaje. Cuando una aplicación recibe un mensaje de objeto, el cuerpo del mensaje es de solo lectura. Si una aplicación llama al método `Clear Body` de la interfaz IMessage para .NET cuando el cuerpo de un mensaje de objeto es de sólo lectura, el cuerpo pasa a ser legible y grabable. El método también borra el cuerpo.

Mensajes de corriente de datos

El cuerpo de un mensaje de corriente de datos contiene una secuencia de valores, donde cada valor tiene un tipo de datos asociado.

El tipo de datos de un valor es uno de los tipos de datos XMS listados en Tabla 100 en la página 693.

Después de que una aplicación haya creado un mensaje de corriente de datos, se puede escribir en el cuerpo del mensaje. La aplicación ensambla los datos de aplicación en el cuerpo llamando a los métodos de escritura apropiados de la interfaz IStreamMessage para .NET. Cada vez que la aplicación escribe un valor en la corriente de datos del mensaje, el valor y su tipo de datos se ensamblan inmediatamente después del valor anterior escrito por la aplicación. XMS mantiene un cursor interno para recordar la posición del último valor que se ha ensamblado.

Cuando la aplicación envía el mensaje, el cuerpo del mensaje se convierte en de solo lectura. De esta forma, la aplicación puede enviar el mensaje varias veces.

Cuando una aplicación recibe un mensaje de corriente de datos, el cuerpo del mensaje es de solo lectura. La aplicación puede utilizar los métodos de lectura apropiados de la interfaz IStreamMessage para .NET para leer el contenido de la corriente de datos de mensaje. La aplicación lee los valores en secuencia, y XMS mantiene un cursor interno para recordar la posición del último valor que leído.

Cuando una aplicación lee un valor de la corriente de datos de mensaje, el valor se puede convertir mediante XMS a otro tipo de datos. Por ejemplo, para leer un entero de la corriente de datos de mensaje, una aplicación puede llamar al método ReadString, que devuelve el entero como una serie. Las conversiones soportadas son las mismas que las que están soportadas cuando XMS convierte un valor de propiedad de un tipo a otro. Si desea más información sobre las conversiones soportadas, consulte “Conversión implícita de un valor de propiedad de un tipo de datos a otro” en la página 666.

Si se produce un error mientras una aplicación está intentando leer un valor de la corriente de datos de mensaje, el cursor no avanza. La aplicación puede recuperarse del error intentando leer el valor como otro tipo de datos.

Si una aplicación llama al método Restablecer de la interfaz IStreamMessage para XMS cuando el cuerpo de un mensaje de corriente de datos es de solo escritura, el cuerpo pasa a ser de solo lectura. El método también vuelve a colocar el cursor al inicio de la corriente de datos de mensaje.

Si una aplicación llama al método `Clear Body` de la interfaz IMessage para XMS cuando el cuerpo de un mensaje de corriente es de sólo lectura, el cuerpo pasa a ser de sólo escritura. El método también borra el cuerpo.

Mensajes de texto

El cuerpo de un mensaje de texto contiene una serie.

Después de que una aplicación crea un mensaje de texto, el cuerpo del mensaje se puede leer y escribir. El cuerpo se sigue pudiendo leer y escribir después de que la aplicación envíe el mensaje. Cuando una aplicación recibe un mensaje de texto, el cuerpo del mensaje es de solo lectura. Si una aplicación llama al método Borrar cuerpo de la interfaz IMessage para .NET cuando el cuerpo de un mensaje de texto es de solo lectura, el cuerpo pasa a poderse leer y escribir. El método también borra el cuerpo.

Tipos de datos para elementos de datos de aplicación

Para asegurarse de que una aplicación XMS puede intercambiar mensajes con una aplicación IBM MQ classes for JMS , ambas aplicaciones deben ser capaces de interpretar los datos de aplicación en el cuerpo de un mensaje de la misma forma.

Por este motivo, cada elemento de los datos de aplicación escritos en el cuerpo de un mensaje por una aplicación XMS debe tener uno de los tipos de datos listados en Tabla 100 en la página 693. Para cada tipo de datos, la tabla muestra el tipo de datos Java compatible. XMS proporciona los métodos para escribir elementos de datos de aplicación solo con estos tipos de datos.

Tabla 100. Tipos de datos XMS que son compatibles con tipos de datos Java

Tipo de datos de XMS	**Representa**	**Tipo de datos Java compatible**
System.Boolean	El valor booleano true o false	boolean
System.Char16	Carácter de doble byte	char
System.SByte	Entero de 8 bits firmado	byte
System.Int16	Entero de 16-bits firmado	short
System.Int32	Entero de 32-bits firmado	int
System.Int64	Entero de 64-bits firmado	Entero largo
System.Float	Número de coma flotante firmado	float
System.Double	Número de coma flotante de precisión doble firmado	double
System.String	Serie de caracteres	Serie

Si desea más información sobre el tamaño, el valor máximo y el valor mínimo de cada uno de estos tipos de datos, consulte “Tipos primitivos de XMS” en la página 666.

Selectores de mensaje

Una aplicación XMS utiliza selectores de mensajes para seleccionar los mensajes que desea recibir.

Cuando una aplicación crea un consumidor de mensajes, puede asociar una expresión de selector de mensajes al consumidor. La expresión de selector de mensajes especifica los criterios de selección.

Cuando una aplicación se está conectando al gestor de colas IBM WebSphere MQ 7.0, la selección de mensajes se realiza en el lado del gestor de colas. XMS no realiza ninguna selección y, simplemente, entrega el mensaje que ha recibido del gestor de colas proporcionando, de esta forma, un mejor rendimiento.

Una aplicación puede crear más de un consumidor de mensajes, cada uno con su propia expresión de selector de mensajes. Si un mensaje entrante cumple los criterios de selección de más de un consumidor de mensajes, XMS entrega el mensaje a cada uno de estos consumidores.

Una expresión de selector de mensajes puede hacer referencia a las propiedades de mensaje siguientes:

- Propiedades definidas por JMS
- Propiedades definidas por IBM
- Propiedades definidas por aplicación

También puede hacer referencia a los campos de cabecera de mensaje siguientes:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Sin embargo, una expresión de selector de mensajes no puede hacer referencia a los datos del cuerpo de un mensaje.

A continuación, se muestra un ejemplo de expresión de selector de mensajes:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS entrega un mensaje a un consumidor de mensajes con esta expresión de selector de mensajes solo si el mensaje tiene una prioridad mayor que 3; una propiedad definida por la aplicación, manufacturer (fabricante), con un valor `Jaguar;` y otra propiedad definida por la aplicación, model (modelo), con un valor de `xj6` o `xj12.`

Las reglas de sintaxis para formar una expresión de selector de mensajes en XMS son las mismas que las de IBM MQ classes for JMS. Si desea más información sobre cómo construir una expresión de selector de mensajes, consulte la documentación del producto IBM MQ, Tenga en cuenta que, en una expresión de selector de mensajes, los nombres de las propiedad definidas por JMS deben ser los nombres JMS y los nombres de las propiedades definidas por IBM deben ser los nombres IBM MQ classes for JMS. No puede utilizar los nombres XMS en una expresión de selector de mensajes.

Correlación de mensajes XMS en mensajes IBM MQ

Los campos de cabecera JMS y las propiedades de un mensaje XMS se correlacionan en las estructuras de cabecera de un mensaje IBM MQ.

Cuando una aplicación XMS está conectada a un gestor de colas IBM MQ, los mensajes enviados al gestor de colas se correlacionan en mensajes IBM MQ de la misma forma que los mensajes IBM MQ classes for JMS se correlacionan en mensajes IBM MQ en circunstancias similares.

Si la propiedad XMSC_WMQ_TARGET_CLIENT de un objeto Destination está establecida en XMSC_WMQ_TARGET_DEST_JMS, los campos de cabecera JMS y las propiedades de un mensaje enviado al destino se correlacionan en campos de las estructuras de cabecera MQMD y MQRFH2 del mensaje IBM MQ. Establecer la propiedad XMSC_WMQ_TARGET_CLIENT de esta forma da por supuesto que la aplicación que recibe el mensaje puede manejar una cabecera MQRFH2. Por lo tanto, la aplicación receptora podría ser otra aplicación XMS, una aplicación IBM MQ classes for JMS o una aplicación IBM MQ nativa que se ha diseñado para manejar una cabecera MQRFH2.

Si la propiedad XMSC_WMQ_TARGET_CLIENT de un objeto Destination está establecida en XMSC_WMQ_TARGET_DEST_MQ en su lugar, los campos de cabecera JMS y las propiedades de un mensaje enviado al destino se correlacionan en los campos de la estructura de cabecera MQMD del mensaje IBM MQ. El mensaje no contiene una cabecera MQRFH2, y se ignora cualquier campo de cabecera JMS y propiedad que no se pueda correlacionar en los campos de la estructura de cabecera MQMD. Por lo tanto, la aplicación que recibe el mensaje puede ser una aplicación IBM MQ nativa que no se ha diseñado para manejar una cabecera MQRFH2.

Los mensajes IBM MQ recibidos de un gestor de colas se correlacionan en los mensajes XMS de la misma forma que los mensajes IBM MQ se correlacionan en los mensajes IBM MQ classes for JMS en circunstancias similares.

Si un mensaje IBM MQ entrante tiene una cabecera MQRFH2, el mensaje XMS resultante tiene un cuerpo cuyo tipo se determina mediante el valor de la propiedad **Msd** incluida en la carpeta `mcd` de la cabecera MQRFH2. Si la propiedad **Msd** no está presente en la cabecera MQRFH2, o si el mensaje IBM MQ no tiene ninguna cabecera MQRFH2, el mensaje XMS resultante tiene un cuerpo cuyo tipo se determina mediante el valor del campo *`Format`* en la cabecera MQMD. Si el campo *`Format`* está establecido en MQFMT_STRING, el mensaje XMS es un mensaje de texto. De lo contrario, el mensaje XMS es un mensaje de bytes. Si el mensaje IBM MQ no tiene ninguna cabecera MQRFH2, solo se establecen los campos de cabecera JMS y las propiedades que se pueden derivar de campos de la cabecera MQMD.

Para obtener más información sobre cómo correlacionar mensajes IBM MQ classes for JMS en mensajes IBM MQ, consulte "Correlación de mensajes JMS en mensajes IBM MQ" en la página 154.

Lectura y escritura del descriptor de mensaje desde una aplicación IBM MQ Message Service Client (XMS) for .NET

Puede acceder a todos los campos del descriptor de mensaje (MQMD) en el mensaje IBM MQ excepto StrucId y Version; BackoutCount se puede leer pero no grabar.

Los atributos de mensaje proporcionados por IBM MQ Message Service Client (XMS) for .NET facilitan aplicaciones XMS para establecer campos MQMD y, también, para dirigir aplicaciones IBM WebSphere MQ.

Se aplican algunas restricciones al utilizar la mensajería de publicación/suscripción. Por ejemplo, campos MQMD como MsgID y CorrelId, si están establecidos, se ignoran.

La función tampoco está disponible cuando la propiedad **`PROVIDERVERSION`** se establece en 6.

Acceso a datos de mensaje de IBM MQ desde una aplicación IBM MQ Message Service Client (XMS) for .NET

Puede acceder a los datos completos de mensaje de IBM MQ incluyendo la cabecera MQRFH2 (si está presente) y cualquier otra cabecera de IBM MQ (si está presente) en una aplicación IBM MQ Message Service Client (XMS) for .NET como cuerpo de un JMSBytesMessage.

La función descrita en este tema sólo está disponible cuando se conecta a un gestor de colas IBM WebSphere MQ 7.0 o posterior y el proveedor de mensajería de IBM MQ está en modalidad normal.

Las propiedades del objeto Destination determinan cómo la aplicación XMS accede a la totalidad de un mensaje IBM MQ (incluyendo la cabecera MQRFH2, si está presente) como el cuerpo de un JMSBytesMessage.

ALW Desarrollo de aplicaciones cliente AMQP

El soporte de IBM MQ para las API de AMQP permite a un administrador de IBM MQ crear un canal AMQP. Cuando se inicia, este canal define un número de puerto que acepta conexiones procedentes de aplicaciones cliente AMQP.

Puede instalar un canal AMQP en sistemas AIX, Linux, and Windows ; no está disponible en IBM i o z/OS.

Una aplicación cliente AMQP 1.0 puede conectarse al gestor de colas con un canal AMQP.

Desarrollo de aplicaciones utilizando la biblioteca de Apache Qpid JMS

Introducción

La biblioteca de Apache Qpid JMS utiliza el protocolo AMQP 1.0 para proporcionar una implementación de la especificación JMS 2.

Apache Qpid JMS utiliza algunos aspectos del protocolo AMQP 1.0 de una forma diferente de las API de mensajería de MQ Light . IBM MQ 9.2 añade soporte a los canales AMQP de IBM MQ , para que

las aplicaciones Apache Qpid JMS puedan conectarse a IBM MQ y realizar la mensajería de publicación/suscripción, incluido el uso de suscripciones compartidas.

V 9.3.0 IBM MQ 9.3 añade soporte adicional a los canales AMQP de IBM MQ , para que las aplicaciones de Apache Qpid JMS puedan conectarse a IBM MQ y realizar la mensajería punto a punto. Consulte “Soporte punto a punto en canales AMQP” en la página 701 para obtener más información.

V 9.3.0 IBM MQ 9.3.0 añade soporte de examen de cola adicional para canales AMQP de IBM MQ , para que las aplicaciones JMS de Apache Qpid puedan conectarse a IBM MQ y realizar el examen de los mensajes de una cola. Consulte “Soporte punto a punto en canales AMQP” en la página 701 para obtener más información.

V 9.3.0 IBM MQ 9.3.0 añade dos atributos de canal adicionales para canales AMQP, TMPMODEL y TMPQPRFX. Estos atributos son para la cola modelo y el prefijo de cola temporal que se va a utilizar al crear una cola temporal.

Intercomunicación con otras aplicaciones IBM MQ

Es posible enviar mensajes entre aplicaciones Apache Qpid JMS y otras aplicaciones IBM MQ . Por ejemplo, una aplicación Apache Qpid puede publicar mensajes sobre un tema y las aplicaciones MQ Light pueden recibirlos creando una suscripción.

Una aplicación Apache Qpid JMS también puede publicar mensajes consumidos por aplicaciones IBM MQ tradicionales, por ejemplo, utilizando la llamada de API MQSUB para suscribirse al mismo tema.

De forma similar, las aplicaciones Apache Qpid JMS pueden suscribirse a temas de IBM MQ en los que las aplicaciones IBM MQ tradicionales publican mensajes.

También es posible que una aplicación Apache Qpid JMS comparta una suscripción con una aplicación MQ Light , siempre que ambos clientes especifiquen el mismo nombre de compartición y patrón de tema.

Tenga en cuenta que, para hacerlo, la aplicación Apache Qpid JMS no debe conectarse con un ID de cliente. Esto garantiza que el nombre de suscripción de IBM MQ utilizado por ambas aplicaciones sea el mismo.

Atención: No es posible que una aplicación Apache Qpid JMS comparta una suscripción con una aplicación IBM MQ JMS .

Restricciones deApache Qpid JMS

Se da soporte a las siguientes prestaciones de JMS :

- Modalidad de acuse de recibo de cliente, acuse de recibo automático y dups ok (DUPS_OK_RECONOCER)
  - Conexión con o sin credenciales
  - Creación de un consumidor en un destino de tema
  - Creación de un consumidor duradero en un destino de tema
  - Creación de un consumidor compartido en un destino de tema
  - Creación de un consumidor duradero compartido en un destino de tema
  - Modalidades de acuse de recibo del cliente y de acuse de recibo automático
  - Acuse de recibo de mensaje y acuse de recibo de sesión
  - Anulación de la suscripción de una suscripción duradera
  - V 9.3.0 Creación de una cola temporal
  - V 9.3.0 Creación de un consumidor en una cola o un destino de cola temporal
  - V 9.3.0 MessageListeners de JMS

  - V 9.3.0 Consumidor JMS para recibir el cuerpo; el método JMS 2.0 denominado `Consumer.receiveBody()`
  - V 9.3.0 Se da soporte a los siguientes tipos de mensajes JMS:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - V 9.3.0 Examinar mensajes de una cola

Las siguientes prestaciones de JMS no están soportadas por los clientes AMQP:

- El uso de sesiones con transacción y JMSContexts con transacción
  - El uso de selectores de mensajes
  - El uso del atributo **nolocal**
  - El uso de sesiones con transacción
  - El uso del retardo de entrega
  - En IBM MQ 9.3.0, examinar mensajes de una cola.
  - Creación de varias suscripciones duraderas o consumidores con el mismo ID de cliente y tema
  - V 9.3.0 Temas temporales JMS
  - Los filtros AMQP no están soportados.

V 9.3.3 A partir de IBM MQ 9.3.3 , la nota siguiente ya no se aplica a los usuarios de Continuous Delivery .

Atención: V 9.3.0 Acuse de recibo del cliente: si se van a utilizar transferencias de mensajes de AMQP no resueltas, es decir, cuando se requiere un acuse de recibo del cliente de los mensajes, los clientes deben resolver de forma puntual enviando acuses de recibo de mensajes de forma puntual cuando se utilice la modalidad de acuse de recibo del cliente, o bien considerar establecer la propiedad del gestor de colas MARKINT (**MsgMarkBrowseInterval**) en un valor más alto.

El valor predeterminado para **MsgMarkBrowseInterval** es de cinco segundos. Si una aplicación no se establece dentro de este valor predeterminado, es posible que se vean mensajes duplicados. Para evitar la duplicación de mensajes, debe aumentar el valor de **MsgMarkBrowseInterval** en consecuencia, estableciéndolo idealmente en NOLIMIT, para representar un intervalo de tiempo ilimitado. Si una aplicación se bloquea o se desconecta antes de que se resuelva un mensaje, los mensajes se ponen a disposición de otra aplicación.

Consulte **MsgMarkBrowseInterval** para obtener más información. Como se trata de una propiedad de gestor de colas, el valor que establezca se aplicará a todas las aplicaciones conectadas a dicho gestor de colas.

En AMQP, **MsgMarkBrowseInterval** sólo es válido para colas, no para suscripciones.

Descarga de clientes de AMQP de ejemplo

IBM MQ no envía clientes AMQP, pero puede descargar clientes MQ Light o descargar clientes AMQP de código abierto basados en bibliotecas Qpid de Apache . Para obtener más información, consulte IBM MQ Light y Apache Qpid.

También puede descargar otros clientes de AMQP de código abierto basados en bibliotecas Apache Qpid. Para obtener más información, consulte https://qpid.apache.org/index.html.

Atención: El soporte de IBM no puede proporcionar soporte de configuración o defecto para estos paquetes de cliente, y cualquier pregunta de uso o informe de defecto de código debe dirigirse a los proyectos respectivos.

Despliegue de clientes de AMQP en IBM MQ

Cuando una aplicación está lista para desplegarse, necesita todas las funciones de supervisión, fiabilidad y seguridad de otras aplicaciones empresariales. La aplicación puede también intercambiar datos con otras aplicaciones empresariales.

Cuando haya desplegado un cliente de AMQP, puede intercambiar mensajes con aplicaciones IBM MQ. Por ejemplo, si utiliza el cliente AMQP para enviar un mensaje de serie JavaScript , la aplicación IBM MQ recibe un mensaje MQ , donde el campo de formato del MQMD se establece en MQSTR.

Gestión del canal AMQP

El canal AMQP se puede gestionar del mismo modo que otros canales MQ. Puede utilizar los mandatos de script de WebSphere MQ (MQSC), los mensajes de mandato PCF o IBM MQ Explorer para definir, iniciar, detener y gestionar los canales. En Creación y utilización de canales AMQP se proporcionan mandatos de ejemplo para definir e iniciar la conexión de clientes con un gestor de colas.

Cuando se inicia un canal AMQP, puede probarlo conectando un cliente AMQP 1.0 . Por ejemplo, MQ Light, Apache Qpid Proton o Apache Qpid JMS.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW MQ Light, Apache Qpid JMSy AMQP (Advanced Message Queuing Protocol)

El cliente MQ Light , los clientes Qpid de Apache como Apache Proton y las API Apache Qpid JMS se basan en el protocolo de conexión OASIS Standard AMQP 1.0 . AMQP especifica cómo se envían los mensajes entre los emisores y los receptores. Una aplicación actúa como emisor cuando la aplicación envía un mensaje al intermediario de mensajes como, por ejemplo, IBM MQ. IBM MQ actúa como emisor cuando envía un mensaje a una aplicación AMQP.

Algunas de las ventajas de AMQP son las siguientes:

- Un protocolo estandarizado abierto
- Compatibilidad con otros clientes de AMQP 1.0 de código abierto
- Muchas implementaciones de cliente de código abierto disponibles

Aunque cualquier cliente de AMQP 1.0 puede conectarse a un canal AMQP, algunas características de AMQP no están soportadas, por ejemplo las transacciones o varias sesiones.

Para obtener más información, consulte el sitio web deAMQP.org y OASIS Standard AMQP 1.0 PDF.

Las API MQ Light y Apache Qpid JMS tienen las características de mensajería siguientes:

- Entrega de mensajes una vez como máximo
- Entrega de mensajes una vez como mínimo
- Direccionamiento de destino de serie de tema
- Durabilidad de mensaje y destino
- Destinos compartidos para permitir que varios suscriptores compartan la carga de trabajo
- Toma de control de cliente para una fácil resolución de clientes bloqueados
- Lectura anticipada de mensajes configurable
- Acuse de recibo de mensajes configurable

Para obtener la documentación completa de la API Apache Qpid JMS , consulte Qpid JMS.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Soporte de AMQP 1.0

Los canales AMQP proporcionan un nivel de soporte para las aplicaciones compatibles con AMQP 1.0.

Los canales AMQP dan soporte a un subconjunto del protocolo AMQP 1.0. Puede conectar clientes compatibles con AMQP 1.0 a un canal AMQP de IBM MQ . Para utilizar todas las características de mensajería soportadas por canales AMQP, debe establecer correctamente el valor de determinados campos de AMQP 1.0.

Esta información describe la forma en la que se debe dar formato a los campos AMQP y lista las características de la especificación AMQP 1.0 que no admiten los canales AMQP.

Las características siguientes de la especificación AMQP 1.0 no se admiten o están limitadas en su uso:

Marco ATTACH

V 9.3.0

Los canales AMQP esperan que las prestaciones del marco ATTACH contengan una de las siguientes:

```
topic
temporary queue
queue
shared
```

Las prestaciones implican el tipo de objeto y, en el caso de varias prestaciones, el orden de prioridad de selección de la prestación es tema, cola temporal, cola.

Si una prestación no contiene un valor esperado, la prestación predeterminada es el tema. Las demás prestaciones se ignoran.

Nota: Algunos clientes AMQP no establecen estas prestaciones y obtendrán el comportamiento predeterminado de IBM MQ de publicación/suscripción. Por ejemplo, el conector Quarkus Reactive Messaging AMQP 1.0 sólo establece prestaciones de la versión 2.8.0CR1 en adelante.

V 9.3.0 Los canales AMQP esperan que el `distribution-Mode` en el marco ATTACH contenga uno de los siguientes, para un origen o destino:

- `Move`
- `copiar`

donde `move` implica una obtención destructiva, y `copy` implica navegador.

Nota: Si `distribution-Mode` no se ha establecido, o se ha establecido en otro valor que no sea `copy`, se presupone `move` .

Nombres de enlace

Los canales AMQP esperan que el nombre de un enlace AMQP siga uno de los cinco formatos:

- Un tema simple (para la publicación y suscripción)
  - Publicación de mensajes: una serie de tema simple (por ejemplo, un nombre de enlace de "`/sports/football`") provoca que se publique un mensaje en el tema `/sports/football`.
  - Suscripción a un tema para recibir mensajes: una serie de tema simple (por ejemplo, un nombre de enlace de "`/sports/football`") provoca que se defina una suscripción en el tema `/sports/football`.
- Un tema detallado privado (para la suscripción)

  - Una serie de tema detallada que describe una suscripción privada con el formato: "`private:`*`topic string`*" (por ejemplo: "`private:/sports/football`"). El comportamiento es idéntico a una serie de tema sin formato. La declaración `private` diferencia una suscripción específica a un cliente de AMQP concreto de una suscripción compartida entre clientes.
- Un tema detallado compartido (para la suscripción)
  - Una serie de tema detallada que describe una suscripción compartida con el formato: "`share:`*`share name`*`:`*`topic string`*" (por ejemplo: "`share:bbc:/sports/football`").
- V 9.3.0 Una cola (para mensajería punto a punto para productor y consumidor)
  - Productor para enviar mensajes; una serie de nombre de cola hace que un productor envíe un mensaje en una cola.
  - Consumidor para recibir mensajes; una serie de nombre de cola hace que un consumidor reciba mensajes de una cola.
- V 9.3.0 En blanco (para la mensajería de punto a punto en una cola temporal)
  - Productor para enviar mensajes en una cola temporal; en blanco hace que un productor envíe un mensaje en una cola temporal.
  - Consumidor para recibir mensajes en una cola temporal; en blanco hace que un consumidor reciba mensajes de una cola temporal.

Para obtener más información sobre cómo se correlacionan los mensajes AMQP con y desde los mensajes IBM MQ , consulte “Correlación entre campos de AMQP y campos de IBM MQ (mensajes entrantes)” en la página 705.

Longitudes máximas para series de nombres, nombres compartidos e ID de cliente

La serie de tema, el nombre compartido y el ID de cliente deben estar incluidos en 10237 bytes. Además, la longitud máxima de un ID de cliente es 256 caracteres.

Estas longitudes máximas significan que puede tener lo siguiente:

- una serie de tema muy larga, siempre que el nombre compartido sea corto
- un nombre compartido largo, pero una serie de tema corta

ID de contenedor

Los canales AMQP esperan que el ID de contenedor de una ejecución de AMQP Open contenga un ID de cliente AMQP exclusivo. La longitud máxima de un ID de cliente AMQP es de 256 caracteres y el ID puede contener caracteres alfanuméricos, signo de porcentaje (%), barra inclinada (/), punto (.) y subrayado (_).

Sesiones

Los canales AMQP solo dan soporte a una única sesión AMQP. Un cliente de AMQP que intenta crear más de una sesión AMQP recibe un mensaje de error y se desconecta del canal.

Transacciones

Los canales AMQP no dan soporte a transacciones AMQP. Un marco adjunto de AMQP que intenta coordinar una nueva transacción o un marco de transferencia de AMQP que intenta declarar una nueva transacción se rechazará con un mensaje de error.

Estado de entrega

Los canales AMQP sólo dan soporte a un estado de entrega para tramas de eliminación de Aceptado, Liberado o Modificado. Tenga en cuenta que, cuando se utiliza un estado de Modified, los canales AMQP no dan soporte a la opción undeliverable-here.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

V 9.3.0 ALW Soporte punto a punto en canales AMQP

El canal AMQP de IBM MQ proporciona soporte para enviar mensajes a colas y recibir mensajes de colas.

Los clientes AMQP como una biblioteca Apache Qpid™ JMS solicitan una prestación `queue` o `temporary-queue` al enviar la trama de conexión AMQP. Las prestaciones permiten al canal AMQP identificar el objeto como cola, cola temporal o tema. En ausencia de una prestación de cola o de cola temporal, o incluso de cualquiera de las prestaciones, se presupone que la solicitud es para un tema.

Los canales AMQP de IBM MQ proporcionan soporte de tipo de cola para lo siguiente:

Recepción y envío de cola
Los mensajes se pueden enviar a una cola y consumir desde una cola. Para consumir mensajes, se da soporte a las modalidades síncrona y asíncrona.

V 9.3.0 **Mensaje de examen de cola**
Además de transferir mensajes a una cola y obtener mensajes de una cola, los mensajes también se pueden examinar desde una cola.

Soporte de cola temporal
Los mensajes pueden enviarse a una cola temporal y consumirse desde una cola temporal. Tenga en cuenta que la supresión de cola temporal está soportada, si el mismo objeto de cola temporal utilizado para crear la cola temporal también se utiliza para suprimir la cola temporal.

El sistema SYSTEM.DEFAULT.MODEL.QUEUE se utiliza al crear una cola temporal y el prefijo para la cola temporal será `AMQP.*`.

El sistema SYSTEM.DEFAULT.MODEL.QUEUE es de forma predeterminada una cola dinámica temporal, pero puede utilizar la propiedad **`Definition type`** en SYSTEM.DEFAULT.MODEL.QUEUE para cambiar la cola para que sea una cola dinámica permanente.

cola dinámica permanente
Una cola dinámica permanente se suprime cuando un cliente AMQP, como una biblioteca Apache Qpid JMS , envía una solicitud con una trama `detach` con el atributo **`closed`** establecido en *true*.

Importante:

Comportamiento de Qpid JMS :
Debe llamar a un mandato de la API JMS de Qpid, por ejemplo, el método `javax.jms.TemporaryQueue.delete()` para destruir la cola después de su uso y este proceso también borra los mensajes presentes en la cola.

Si no emite un mandato de este tipo, la cola permanece con los mensajes todavía presentes, cuando se cierra la conexión.

-

cola dinámica temporal
Se suprime una cola dinámica temporal cuando el cliente AMQP cierra la conexión.

Importante:

Comportamiento de Qpid JMS :
Si llama a un mandato de la API JMS de Qpid, por ejemplo, el método `javax.jms.TemporaryQueue.delete()` , cierra la conexión JMS o la conexión se interrumpe, la cola se suprime y los mensajes se pierden.

El cierre de una sesión de JMS en sí mismo no hace que se suprima la cola temporal, aunque la cola temporal se haya podido crear utilizando el método `javax.jms.Session.createTemporaryQueue()` .

-

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Correlación de campos de mensaje AMQP e IBM MQ

Los mensajes de AMQP constan de una cabecera, anotaciones de entrega, anotaciones de mensajes, propiedades, propiedades de aplicación, cuerpo y pie de página.

Los mensajes de AMQP constan de las partes siguientes:

Cabecera

La cabecera opcional contiene cinco atributos fijos del mensaje:

- **durable** - especifica requisitos de durabilidad
- **priority** - prioridad relativa del mensaje
- **ttl** - tiempo de vida en milisegundos
- **first-acquirer** - si su valor es true, el mensaje no ha sido adquirido por ningún otro enlace
- **número-entrega** - número de intentos de entrega anteriores no satisfactorios.

Anotaciones de entrega

Opcional. Especifica atributos de cabecera no estándar del mensaje para distintos destinatarios. Las anotaciones de entrega transmiten información desde el nodo de emisión al nodo de recepción.

Anotaciones de mensajes

Opcional. Especifica atributos de cabecera no estándar del mensaje para distintos destinatarios. La sección de anotaciones de mensajes se utiliza para las propiedades del mensaje destinadas a la infraestructura y se deben propagar a través de cada paso de entrega.

Propiedades

Opcional. Esta parte es equivalente al descriptor de mensaje de MQ. Contiene los campos fijos siguientes:

- **message-id** - identificador de mensaje de aplicación
- **user-id** - ID de creación del usuario
- **to** - dirección del nodo de destino del mensaje
- **subject** - asunto del mensaje
- **reply-to** - nodo al que se envía la respuesta
- **correlation-id** - identificador de correlación de aplicaciones
- **content-type** - tipo de contenido MIME
- **content-encoding** - tipo de contenido MIME. Se utiliza como modificador de content-type.
- **absolute-expiry-time** - fecha y hora en que el mensaje se considera caducado
- **creation-time** - fecha y hora en que se creó el mensaje
- **group-id** - grupo al que pertenece el mensaje
- **group-sequence** - número de secuencia del mensaje dentro de su grupo
- **reply-to-group-id** - grupo al que pertenece el mensaje de respuesta

Propiedades de aplicación

Equivalente a las propiedades del mensaje MQ.

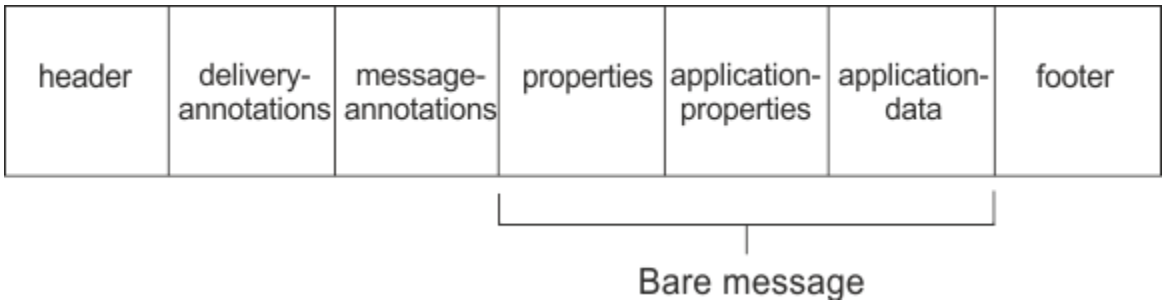
Cuerpo

Equivalente a la carga útil de usuario de MQ.

Pie de página

Opcional. El pie de página se utiliza para detalles sobre el mensaje o la entrega que sólo se puede calcular o evaluar después de que se haya construido o visto el mensaje básico completo (por ejemplo, hashes de mensajes, HMACs, firmas y detalles de cifrado).

El formato del mensaje de AMQP se ilustra en la figura siguiente:



Las parte correspondiente a las propiedades, propiedades de aplicación y datos de aplicación se conoce como "mensaje básico". Este es el mensaje tal como es enviado por el remitente, y es inmutable. El destinatario ve el mensaje completo, incluidos la cabecera, el pie de página, las anotaciones de entrega y las anotaciones de mensajes.

Para obtener una descripción completa del formato de mensaje de AMQP 1.0, consulte el estándar OASIS en https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Correlación de campos de IBM MQ con campos de AMQP (mensajes salientes)

Cuando se publica un mensaje de IBM MQ y IBM MQ lo envía a un consumidor AMQP, propagará algunos de los atributos del mensaje de IBM MQ en los atributos de mensaje AMQP equivalentes.

cabecera

Solo se incluye una cabecera si uno de los cinco campos de la cabecera contiene un valor no predeterminado. En la cabecera solo se incluyen los campos cuyo valor es distinto del valor predeterminado. Los cinco campos de cabecera se obtienen inicialmente de la propiedad equivalente `mq_amqp.Hdr`, si está establecida, y luego se modifican de acuerdo con la tabla siguiente:

Tabla 101. Correlaciones de campos de cabecera

Campo	Valor predeterminado	Valor
durable	falso	True si `MQMD.Persistence` se establece en `MQPER_PERSISTENT`, de lo contrario, false.
priority	4	Se obtiene de `mq_amqp.Hdr.Pri`, si está establecido, en otro caso se obtiene de `MQMD.Priority`, si está establecido. Si ninguno de los dos está establecido, se establece en 4.
ttl	n/d	`MQMD.Expiry` en milisegundos. Si el valor de `MQMD.Expiry` es MQEI_UNLIMITED, se establece en el valor máximo del campo AMQP ttl
first-acquirer	falso	Se obtiene de `mq_amqp.Hdr.Fac`, si está establecido, en caso contrario es falso.
delivery-count	0	Se obtiene de `mq_amqp.Hdr.Dct`, si está establecido, en caso contrario es 0.

delivery-annotation

Se establece según sea necesario mediante el canal AMQP.

message-annotation

No incluido.

Propiedades

Las **propiedades** provienen inalteradas de las propiedades equivalentes `mq_amqp.Prp`, si están establecidas. Si el mensaje no era originalmente un mensaje AMQP (es decir, PutApplType no es MQAT_AMQP), se genera una sección de propiedades tal como se describe en la tabla siguiente:

Tabla 102. Correlaciones de campos de propiedades

Nombre	**Valor**
message-id	El campo `MQMD.MsgId` se establece como binario.
id-usuario	El formato UTF-8 de `MQMD.UserIdentifier` se establece como binario en orden de bytes de la red.
por:	La cola de la que se obtuvo el mensaje, o, para una publicación, la serie de tema.
subject	No establecido.
respuestas	El campo `MQMD.ReplyToQ` si no está en blanco, en caso contrario no se establece.
correlation-id	El campo `MQMD.CorrelId` se establece como binario si no está en blanco, en caso contrario no se establece.
content-type	No establecido.
content-encoding	No establecido.
absolute-expiry-time	No establecido.
creation-time	Los campos `MQMD.PutDate` y `MQMD.PutTime` se utilizan para generar una indicación de fecha y hora.
group-id	No establecido.
group-sequence	No establecido.
reply-to-group-id	No establecido.

application-properties

Todas las propiedades de IBM MQ contenidas en el grupo "usr" se añaden como propiedades de aplicación (**application-properties**).

cuerpo

El canal AMQP realiza una operación get con conversión para convertir la carga útil de IBM MQ a UTF-8.

Si la carga útil de IBM MQ no contiene un mensaje AMQP, la carga útil de IBM MQ se establece en el cuerpo como una sección de datos de tipo serie para el formato MQFMT_STRING (si la conversión a UTF-8 ha sido satisfactoria), o como una sección de datos binarios.

Si está incluido un mensaje de formato AMQP, este se establece como el cuerpo. Cualquier cabecera de IBM MQ (sin incluir las propiedades de mensajes que se devuelven en un descriptor de contexto de

mensaje) que preceden al mensaje AMQP se agregan al inicio como valor binario si el cuerpo es una secuencia AMQP. De lo contrario, las cabeceras de IBM MQ se descartan.

pie

No se incluye ningún pie.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

Referencia relacionada

MQMD - Descriptor de mensaje

ALW Correlación entre campos de AMQP y campos de IBM MQ (mensajes entrantes)

Cuando el canal AMQP recibe un mensaje y lo coloca en IBM MQ, propaga algunos de los atributos del mensaje AMQP a los atributos del mensaje de IBM MQ equivalente.

Se aplican las restricciones siguientes cuando se correlaciona un mensaje AMQP entrante:

- Si el campo `message-id` o `correlation-id` en la parte de propiedades es un uuid o un ulong, el mensaje se rechaza.
- `message-annotations` hace que se rechace el mensaje.
- Se permiten secciones `delivery-annotations` y `footer`, pero no se propagan al mensaje de IBM MQ.

En las siguientes subsecciones, se muestra la expresión de IBM MQ de un mensaje AMQP.

Descriptor de mensaje

Tabla 103. Descriptor de mensaje para un mensaje AMQP

Campo	Valor
StrucId	MQMD_STRUC_ID
Versión	MQMD_VERSION_1
Informe	MQRO_NONE
MsgType	MQMT_DATAGRAM
Caducidad	Valor obtenido del campo `ttl` en la cabecera de mensaje AMQP
Comentarios	MQFB_NONE
Encoding	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Formato	Véase Carga útil
Prioridad	Valor obtenido del campo `priority` en la cabecera de mensaje AMQP. Si se establece, limitado a un máximo de 9. Si no se establece, toma el valor predeterminado de 4.
Persistence	Si el campo `durable` en la cabecera de mensaje AMQP se establece en true, establézcalo en MQPER_PERSISTENT. De lo contrario, establézcalo en MQPER_NOT_PERSISTENT.
MagId	El gestor de colas asigna un MsgId de 24 bytes exclusivo.

Tabla 103. Descriptor de mensaje para un mensaje AMQP (continuación)	
Campo	**Valor**
CorrelId	Valor obtenido del campo `correlation-id` en las propiedades AMQP, si se establece. Establézcalo en un valor binario de 24 bytes. De lo contrario, establézcalo en MQCI_NONE/.
BackoutCount	0
ReplyToQ	V 9.3.0 Valor tomado del campo `reply-to` en las propiedades AMQP, si se ha establecido. De lo contrario, establézcalo en "".
ReplyToQMgr	""
V 9.3.0 Informe	Valor derivado de las propiedades de informe JMS IBM establecidas en las propiedades de la aplicación AMQP.
UserIdentifier	Establézcalo en el identificador del usuario autenticado que está conectado al canal AMQP
AccountingToken	MQACT_NONE
ApplIdentityData	Serie hexadecimal. Establézcalo en los últimos 8 bytes del identificador de conexión MQ del canal AMQP.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	Valor obtenido del campo `creation-time` de las propiedades AMQP, si se establece. De lo contrario, establézcalo en la fecha actual.
PutTime	Valor obtenido del campo `creation-time` de las propiedades AMQP, si se establece. De lo contrario, establézcalo en la hora actual.
ApplOriginData	""

Propiedades del mensaje

Existen dos razones para establecer las propiedades del mensaje:

- Para permitir que las partes del mensaje AMQP fluyan a través del gestor de colas sin afectar a la carga útil del mensaje.
- Para permitir la selección de `application-properties`.

En la tabla siguiente, se muestran las propiedades que se establecen en el mensaje AMQP:

Tabla 104. Propiedades de mensaje AMQP			
Nombre de propiedad	**Nombre de MQRFH2**	**Tipo**	**Descripción**
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	Una serie de identificación para el canal AMQP. Se utiliza para generar el mensaje, para que las partes interesadas puedan identificar qué versión ha colocado el mensaje (por ejemplo, el equipo de servicio cuando diagnostican problemas). El valor no está validado por el gestor de colas y no debe documentarse externamente.

Tabla 104. Propiedades de mensaje AMQP (continuación)

Nombre de propiedad	Nombre de MQRFH2	Tipo	Descripción
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	La versión del mensaje AMQP. Si no está presente, se supone "1.0". El valor no está validado por el gestor de colas.
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	Una serie de identificación para la API. Se utiliza para enviar el mensaje AMQP al canal, para que las partes interesadas puedan identificar qué versión ha colocado el mensaje (por ejemplo, el equipo de servicio cuando diagnostican problemas). El valor no está validado por el gestor de colas y no debe documentarse externamente.
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	El valor del campo `durable` en la cabecera de mensaje AMQP, si se establece.
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	El valor del campo `priority` en la cabecera de mensaje AMQP, si se establece.
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	El valor del campo `ttl` en la cabecera de mensaje AMQP, si se establece.
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	El valor del campo `first-acquirer` en la cabecera de mensaje AMQP, si se establece.
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	El valor del campo `delivery-count` en la cabecera de mensaje AMQP, si se establece.
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	El valor del campo `message-id` en las propiedades AMQP, si se establece como una serie.
		MQTYPE_BYTE_STRING	El valor del campo `message-id` en las propiedades AMQP, si se establece como una serie de bytes.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	El valor del campo `user-id` en las propiedades AMQP, si se establece.
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	El valor del campo `to` en las propiedades AMQP, si se establece.
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	El valor del campo `subject` en las propiedades AMQP, si se establece.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	El valor del campo `reply-to` en las propiedades AMQP, si se establece.

Tabla 104. Propiedades de mensaje AMQP (continuación)			
Nombre de propiedad	**Nombre de MQRFH2**	**Tipo**	**Descripción**
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	El valor del campo `correlation-id` en las propiedades AMQP, si se establece como una serie.
		MQTYPE_BYTE_STRING	El valor del campo `correlation-id` en las propiedades AMQP, si se establece como una serie de bytes.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	El valor del campo `content-type` en las propiedades AMQP, si se establece.
AMQPContentEncoding	mq_amqp.Prp.Cne	MQTYPE_STRING	El valor del campo `content-encoding` en las propiedades AMQP, si se establece.
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	El valor del campo `absolute-expiry-time` en las propiedades AMQP, si se establece.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	El valor del campo `creation-time` en las propiedades AMQP, si se establece.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	El valor del campo `group-id` en las propiedades AMQP, si se establece.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	El valor del campo `group-sequence` en las propiedades AMQP, si se establece.
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	El valor del campo `reply-to-group-id` en las propiedades AMQP, si se establece.

Cada una de las propiedades de aplicación del mensaje AMQP se establece como una propiedad de mensaje de IBM MQ. La sección `application-properties` debe reconstituirse de forma idéntica byte a byte, por lo que se aplican las siguientes restricciones:

- Si una propiedad de aplicación es rechazada por el código de validación MQSETMP, el mensaje se rechaza. Por ejemplo:
  - El nombre de propiedad tiene una longitud limitada a MQ_MAX_PROPERTY_NAME_LENGTH.
  - El nombre de propiedad debe seguir las reglas definidas por Java Language Specification for Java Identifiers.
  - El nombre de propiedad no debe empezar por `JMS` o `usr.JMS`, excepto las propiedades JMS documentadas que se pueden establecer.
  - El nombre de propiedad no puede ser una palabra clave de SQL.
- Una propiedad de aplicación que contiene el carácter Unicode U+002E (".") hace que el mensaje se rechace. La propiedad debe poder expresarse en el grupo "usr" de propiedades utilizadas por JMS.
- Solo se da soporte a las propiedades de tipo null, boolean, byte, short, int, long, float, double, binary y string. Una propiedad de aplicación con otro tipo hará que el mensaje se rechace.

V 9.3.0 Puede establecer las siguientes propiedades JMS utilizando `application-properties`:

- JMS_IBM_REPORT_EXCEPTION

- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_REPORT_PASS_CORREL_ID
- JMS_IBM_REPORT_DISCARD_MSG

Tenga en cuenta que los nombres y valores de propiedad son coherentes con los detalles de "Correlación de campos específicos del proveedor JMS" en la página 165 equivalentes y que los valores que no son válidos se ignoran.

Carga útil

- Para un AMQP body con una única sección de datos binarios, los datos binarios (excluyendo los bits AMQP) se colocan como carga útil de IBM MQ , con un formato de MQFMT_NONE.
- Para un AMQP body con una sección de datos de serie única, los datos de serie (excluidos los bits AMQP) se colocan como carga útil de IBM MQ , con un formato de MQFMT_STRING.
- De lo contrario, AMQP body forma la carga útil tal cual, con un formato de MQFMT_AMQP.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Fiabilidad de entrega de mensajes

En esta sección se comparan las características de fiabilidad para la API de MQ Light y Apache Qpid JMS.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Fiabilidad de mensajes de MQ Light

Existen cuatro características de la API de MQ Light que le permiten controlar la fiabilidad de la entrega de mensajes a y desde aplicaciones AMQP.

Son las siguientes:

- "Calidad de servicio (QOS) de mensajes" en la página 709
- "Confirmación automática del suscriptor" en la página 710
- "Tiempo de vida de suscripción" en la página 710
- "Persistencia de los mensajes" en la página 711

Calidad de servicio (QOS) de mensajes

La MQ Light API ofrece dos calidades de servicio:

- Como máximo una vez
- Al menos una vez

Puede elegir la calidad de servicio que desea que utilicen los publicadores y suscriptores.

Si está utilizando un cliente de MQ Light, establezca la opción de cliente **qos** o de suscripción en *QOS_AT_MOST_ONCE* o *QOS_AT_LEAST_ONCE*.

Si utiliza un cliente AMQP distinto, establezca el atributo **settled** de la trama de transferencia (para los publicadores) o la trama de eliminación (para los suscriptores) en *true* o *false*, dependiendo de la calidad de servicio que desee.

La calidad de servicio determina cuándo se descarta un mensaje del lado `sending` de una conversación.

Publicación

Si un publicador elige **QOS** *0* (como máximo una vez), el publicador no espera un acuse de recibo del gestor de colas antes de descartar su copia del mensaje.

Si la conexión con el gestor de colas falla antes de que haya terminado el envío, los suscriptores puede que no reciban el mensaje.

Si un publicador elige **QOS** *1* (al menos una vez), el publicador espera a que el gestor de colas acuse recibo de que el mensaje se ha escrito en las colas de suscriptor antes de descartar su copia del mensaje.

Si la conexión con el gestor de colas falla durante el envío, el publicador vuelve a enviar el mensaje cuando se reconecte al gestor de colas.

Suscripción

Si un suscriptor elige **QOS** *0*, el gestor de colas no espera un acuse de recibo del suscriptor antes de descartar su copia del mensaje.

Si la conexión con el suscriptor falla antes de que el suscriptor haya recibido el mensaje, es posible que se pierda el mensaje.

Si un suscriptor elige **QOS** *1*, el gestor de colas espera un acuse de recibo del suscriptor antes de descartar su copia del mensaje. V 9.3.3 Desde IBM MQ 9.3.3, los mensajes reconocidos se eliminan en lotes para mejorar el rendimiento. Para obtener más información, consulte “Eliminación de mensajes AMQP reconocidos de la cola en lotes” en la página 712.

Si la conexión con el suscriptor falla antes de que el suscriptor haya recibido el mensaje, el gestor de colas mantiene el mensaje. El gestor de colas vuelve a enviar el mensaje al suscriptor cuando se reconecta, o a otro suscriptor si se comparte la suscripción.

Confirmación automática del suscriptor

Si un suscriptor elige **QOS** *1* (al menos una vez), debe acusar recibo de cada mensaje antes de que el gestor de colas descarte su copia. El suscriptor puede decidir cuándo acusa recibo de los mensajes.

Con **auto-confirm** establecido en *true*, el cliente MQ Light acusa recibo automáticamente de la entrega de cada mensaje, una vez que el cliente ha recibido correctamente el mensaje a través de la red.

Esto garantiza que si hay un error de red, el mensaje se vuelva a enviar a la aplicación. Sin embargo, todavía es posible que la aplicación pierda el mensaje, si la aplicación falla mientras el cliente de MQ Light está realizando el acuse de recibo del mensaje, y la aplicación lo está procesando.

Con **auto-confirm** establecido en *false*, el cliente MQ Light no acusa recibo automáticamente de la entrega del mensaje, sino que deja que la aplicación decida cuándo se debe acusar recibo.

Esto permite a la aplicación realizar una actualización en un recurso externo, por ejemplo, una base de datos o un archivo, antes de acusar recibo al gestor de colas de que el mensaje se ha procesado y se puede descartar.

Tiempo de vida de suscripción

Cuando se suscribe una aplicación, elige si la suscripción y el destino donde se almacenan los mensajes para dicha suscripción siguen existiendo cuando la aplicación se desconecta.

La MQ Light opción de suscripción **ttl** se utiliza para especificar el tiempo (en milisegundos) que una suscripción sigue existiendo después de que la aplicación se desconecte. Si la aplicación se vuelve a conectar antes de este tiempo, la suscripción se reanuda y la aplicación puede continuar consumiendo mensajes de dicha suscripción.

Si el periodo de tiempo de vida transcurre sin que se reconecte la aplicación, la suscripción se elimina y se pierden todos los mensajes almacenados en su destino, aunque sean mensajes persistentes.

Si es importante no perder mensajes, debe especificar un valor de tiempo de vida en la aplicación que sea lo suficientemente alto para asegurarse de que no se pierdan los mensajes durante una parada.

Persistencia de los mensajes

La persistencia de los mensajes se controla mediante las aplicaciones de publicación y suscripción y la configuración de los objetos de tema de IBM MQ.

Si el suscriptor de AMQP utiliza **QOS** *0* (como máximo una vez) y crea una suscripción no duradera, el canal AMQP siempre coloca mensajes no persistentes en la cola de suscriptor, independientemente de las otras opciones que se describen en el texto siguiente.

Tenga en cuenta que, si el gestor de colas se detiene, se perderán tanto la suscripción como los mensajes.

Si un publicador de AMQP establece la cabecera AMQP **durable** en *true*, el canal AMQP coloca mensajes persistentes en las colas de suscriptor.

Si el gestor de colas se detiene por alguna razón, los mensajes siguen estando disponibles para los suscriptores cuando se reinicia el gestor de colas.

Si no se establece la cabecera **durable**, el canal AMQP elige la persistencia de los mensajes publicados basándose en el atributo **DEFPSIST** del objeto de tema de IBM MQ relevante.

De forma predeterminada, es SYSTEM.BASE.TOPIC, que utiliza un atributo **DEFPSIST** igual a *NO* (no persistente).

Atención: Las versiones posteriores del cliente MQ Light no admiten establecer la cabecera duradera de AMQP.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Fiabilidad de mensajes de Apache Qpid JMS

Existen cuatro características de la biblioteca Apache Qpid™ JMS que le permiten controlar la fiabilidad de la entrega de mensajes a y desde aplicaciones AMQP.

Estos son para:

- “Publicación” en la página 711 V 9.3.0 /Producer para mensajería punto a punto
  - Caducidad del mensaje
  - Persistencia de los mensajes
- “Suscripción” en la página 712
  - Durabilidad de suscripción
  - Modalidad de acuse de recibo de sesión V 9.3.0 (Aplicable también para mensajería de punto a punto de consumidor)

Publicación

Caducidad del mensaje

El establecimiento del valor de tiempo de vida del productor de JMS afecta al tiempo de caducidad asignado a los mensajes publicados por dicho productor de mensajes.

Asegúrese de que el valor de tiempo de vida de un productor de JMS sea lo suficientemente grande como para que los mensajes se consuman antes de que caduquen.

De forma alternativa, dejar el valor de tiempo de vida sin establecer impide que el mensaje caduque en la cola de suscripción.

Persistencia de los mensajes

Si se establece la modalidad de entrega del productor de mensajes JMS , se establece la persistencia del mensaje IBM MQ publicado en el tema especificado.

Asegúrese de que utiliza **DeliveryMode**.PERSISTENT para los mensajes que deben retenerse cuando un gestor de colas finaliza o tiene una parada.

Suscripción

Durabilidad de suscripción

Los canales AMQP dan soporte a la creación de suscripciones duraderas utilizando las versiones duraderas de los métodos de creación de consumidor de JMS :

- **createDurableConsumer()**
- **createSharedDurableConsumer()**

Modalidad de acuse de recibo de sesión

Para garantizar que un mensaje consumido se ha procesado completamente antes de que se elimine de la cola de suscripción de IBM MQ , cree una sesión de JMS utilizando **Session**.La modalidad CLIENT_RECONOCER y utilice el método **message.acknowledge()** para reconocer este mensaje y cualquier otro recibido anteriormente en esta sesión.

Conceptos relacionados

Desarrollo de aplicaciones cliente AMQP
El soporte de IBM MQ para las API de AMQP permite a un administrador de IBM MQ crear un canal AMQP. Cuando se inicia, este canal define un número de puerto que acepta conexiones procedentes de aplicaciones cliente AMQP.

V 9.3.3 Eliminación de mensajes AMQP reconocidos de la cola en lotes

Si una aplicación AMQP utiliza la entrega de mensajes QOS_AT_LEAST_ONCE (1), el servicio AMQP espera un acuse de recibo de la aplicación antes de descartar la copia de un mensaje que conserva después de enviar dicho mensaje a la aplicación. A partir de IBM MQ 9.3.3, los mensajes que se han reconocido se eliminan de la cola en lotes, en lugar de individualmente, lo que mejora el rendimiento.

Acerca de esta tarea

Para Long Term Support y Continuous Delivery antes de IBM MQ 9.3.3, cada mensaje se elimina de la cola individualmente.

A partir de IBM MQ 9.3.3, puede utilizar las dos propiedades del sistema **com.ibm.mq.AMQP.BATCHSZ** y **com.ibm.mq.AMQP.BATCHINT** para ajustar el proceso de confirmaciones en lotes para mejorar el rendimiento:

com.ibm.mq.AMQP.BATCHSZ
: Este atributo define el número máximo de confirmaciones que deben recibirse antes de que el servicio AMQP elimine mensajes. El número puede estar en el rango de 1 a 9999. Si se establece un número no válido, o si el número especificado está fuera de rango, se utiliza el valor predeterminado de 50.

  El tamaño de lote no afecta a la forma en que se transfieren los mensajes. Los mensajes siempre se transfieren individualmente, pero luego se eliminan en un lote después de que el servicio AMQP reciba los acuses de recibo. El tamaño real de un lote puede ser menor que el valor especificado por **com.ibm.mq.AMQP.BATCHINT**. Por ejemplo, un lote se completa si el periodo establecido por el atributo **com.ibm.mq.AMQP.BATCHINT** caduca.

com.ibm.mq.AMQP.BATCHINT
Este atributo define la cantidad de tiempo, en milisegundos, que el servicio AMQP mantiene los mensajes reconocidos en la cola. Si el lote no está lleno, el lote se borra después de esta duración. Puede especificar cualquier número de milisegundos, de 1 a 999.999.999.999. El valor predeterminado es 50. Si no especifica un valor para este atributo, se utiliza el valor predeterminado de 50.

Notas:

1. Si el servicio AMQP espera un acuse de recibo antes de descartar un mensaje depende de cuál de las dos calidades de servicio siguientes está utilizando una aplicación para la entrega de mensajes:
   - QOS para QOS_AT_MOST_ONCE (0)

     Si una aplicación AMQP está utilizando esta calidad de servicio, no reconoce los mensajes, por lo que el servicio AMQP descarta los mensajes después de enviarlos a la aplicación sin esperar un acuse de recibo.
   - QOS_AT_LEAST_ONCE (1)

     Si una aplicación AMQP está utilizando esta calidad de servicio, reconoce los mensajes, por lo que el servicio AMQP conserva una copia de cada mensaje después de enviarlo a la aplicación hasta que recibe un acuse de recibo de la aplicación. Si la aplicación se desconecta de o pierde la conexión antes de reconocer el mensaje, el mensaje se pone a disposición de otras aplicaciones. El servicio AMQP no elimina un mensaje de la cola hasta que se ha reconocido.
2. MQ Appliance Las propiedades del sistema **com.ibm.mq.AMQP.BATCHSZ** y **com.ibm.mq.AMQP.BATCHINT** no son aplicables en IBM MQ Appliance. Se utiliza un valor predeterminado de 50 en IBM MQ Appliance.

.

Procedimiento

Utilice las propiedades del sistema **com.ibm.mq.AMQP.BATCHSZ** y **com.ibm.mq.AMQP.BATCHINT** para ajustar el proceso de confirmaciones en lotes.

A partir de IBM MQ 9.3.3, cuando se crea el gestor de colas, el archivo `amqp_java.properties` contiene los siguientes valores predeterminados para las propiedades del sistema:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

En función de la tasa de mensajes consumida, puede ajustar el proceso de acuses de recibo en lotes para mejorar el rendimiento. Un gestor de colas migrado no tiene estas propiedades en el archivo `amqp_java.properties` . Por lo tanto, para un gestor de colas migrado, o si las propiedades no están establecidas, se utilizan los valores predeterminados. Puede añadir estas propiedades para ajustar los valores para el rendimiento optimizado.

Los mensajes con acuse de recibo se eliminan por lotes cuando se cumple una de las condiciones siguientes:

- El número de mensajes reconocidos alcanza **com.ibm.mq.AMQP.BATCHSZ**.
- **com.ibm.mq.AMQP.BATCHINT** se supera después del inicio del lote.
- La aplicación desconecta o cierra la cola o el tema antes de que se cumplan las dos condiciones anteriores.

ALW Topologías para clientes AMQP con IBM MQ

Topologías de ejemplo para ayudarle a desarrollar los clientes AMQP para que funcionen con IBM MQ.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Clientes de AMQP que se comunican a través de IBM MQ

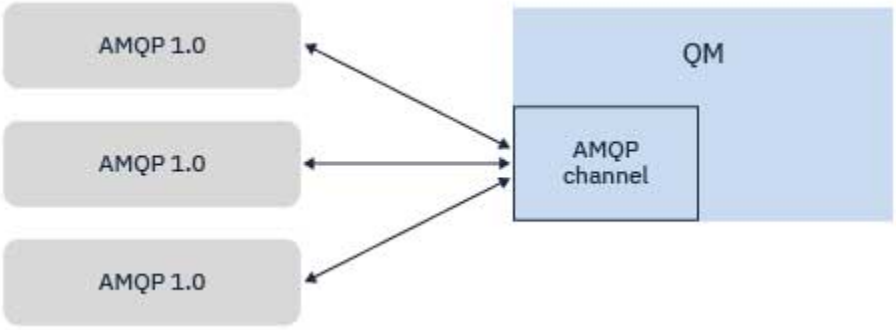
Puede utilizar IBM MQ como proveedor de mensajería para cualquier aplicación que cumpla con AMQP 1.0. Aunque cualquier cliente de AMQP 1.0 puede conectarse a un canal AMQP, algunas características de AMQP no están soportadas, por ejemplo las transacciones o varias sesiones.

Mediante la definición de uno o más canales AMQP, los clientes de AMQP 1.0 se pueden conectar al gestor de colas y enviar mensajes a una serie de tema. Los clientes también se pueden suscribir a un patrón de tema para recibir mensajes que coincidan con el patrón.

En el escenario siguiente, las únicas aplicaciones que envían y reciben mensajes son las aplicaciones AMQP 1.0 .

Las aplicaciones pueden elegir si los destinos creados al suscribirse a una serie de tema son persistentes, de forma que los mensajes no se pierden si la aplicación pierde temporalmente su conexión al gestor de colas.

Las aplicaciones también pueden elegir cómo se conservan los mensajes largos antes de que se depuren en el destino.



Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Clientes de AMQP que intercambian mensajes con aplicaciones IBM MQ

Al definir e iniciar un canal AMQP, las aplicaciones AMQP 1.0 pueden publicar mensajes recibidos por aplicaciones MQ existentes. Los mensajes publicados a través de un canal AMQP se envían todos a temas MQ, no a colas MQ. Una aplicación MQ que ha creado una suscripción utilizando la llamada de API MQSUB recibe los mensajes publicados por las aplicaciones AMQP 1.0, siempre que la serie de tema o el objeto de tema utilizado por la aplicación MQ coincida con la serie de tema publicada por el cliente AMQP.

Los datos, atributos y propiedades de mensaje AMQP se establecen en el mensaje MQ recibido por la aplicación MQ. Para obtener más información sobre las correlaciones de mensajes de AMQP con MQ , consulte “Correlación entre campos de AMQP y campos de IBM MQ (mensajes entrantes)” en la página 705.

Si la aplicación MQ ha creado una suscripción que es duradera, los mensajes publicados por la aplicación AMQP se almacenan en la cola que respalda la suscripción. Los mensajes después son recibidos por la aplicación MQ, cuando la aplicación reanuda su suscripción. Si la aplicación AMQP especifica un tiempo de vida del mensaje y la aplicación MQ no se reconecta dentro de este tiempo de vida, el mensaje caduca de la cola.

Las aplicaciones AMQP 1.0 también pueden consumir mensajes publicados por aplicaciones MQ existentes. Los mensajes publicados por las aplicaciones MQ en un tema MQ o una serie de tema son recibidos por una aplicación AMQP 1.0, siempre que la aplicación se haya suscrito con un patrón de tema que coincida con la serie de tema publicada.

Si la aplicación AMQP 1.0 especifica un valor de tiempo de vida para la suscripción, y la aplicación AMQP se desconecta durante un periodo de tiempo mayor que el tiempo de vida, la suscripción caduca del gestor de colas y los mensajes almacenados en la cola de suscripción se pierden.

Los campos MQMD, las propiedades de mensaje y los datos de aplicación se establecen en el mensaje AMQP recibido por la aplicación AMQP. Para obtener más información sobre las correlaciones de mensajes de MQ con AMQP, consulte "Correlación de campos de IBM MQ con campos de AMQP (mensajes salientes)" en la página 703.

Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

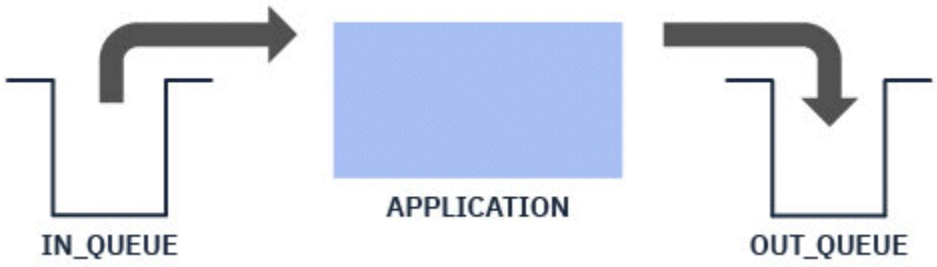
ALW Configuración de clientes de AMQP para interactuar directamente con aplicaciones en colas IBM MQ

V 9.3.0 La implementación AMQP de IBM MQ da soporte a la publicación/suscripción y punto a punto. Para cualquier cliente AMQP que no soporte punto a punto, utilice los pasos siguientes para enviar mensajes a una cola o recibir mensajes de una cola.

Visión general

Por ejemplo, suponga que hay una aplicación que obtiene mensajes de una cola de entrada `IN_QUEUE` y que coloca estos mensajes en una cola de salida `OUT_QUEUE`. Es posible para los clientes de AMQP colocar mensajes en `IN_QUEUE` y obtener mensajes de `OUT_QUEUE`

Nota: No hay ningún cambio necesario en la propia aplicación.



Para que un publicador AMQP coloque un mensaje en una cola, tendrá que crear una suscripción administrativa para la serie de tema en la que está publicando el cliente de AMQP, con un destino de la cola prevista; consulte "Envío de mensajes a la aplicación" en la página 715.

Para que un suscriptor de AMQP obtenga mensajes de una cola, tendrá que sustituir la cola con una cola alias del mismo nombre, con un destino de un objeto de tema que representa la serie de tema a la que está suscrito el cliente de AMQP; consulte "Obtención de mensajes de la aplicación" en la página 716

Envío de mensajes a la aplicación

La aplicación ya está seleccionando mensajes de `IN_QUEUE` y desea que un cliente de AMQP pueda publicar mensajes, así que van a esta cola para que los procese la aplicación.

Para ello, cree una nueva suscripción administrativa, donde la serie de tema de la que recibe los mensajes esta suscripción es la serie de tema en la que publica el cliente de AMQP. La cola de destino de esta suscripción es la cola de entrada para la aplicación, `IN_QUEUE`.

Los mensajes publicados en la serie de tema definida para dicha suscripción administrativa se direccionan al destino definido, en este caso `IN_QUEUE`.

Dando por supuesto que el cliente de AMQP publica en una serie de tema `/application/in`, puede crear una suscripción administrativa `APP_IN`, utilizando el mandato MQSC siguiente:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Cuando haya definido este objeto, todos los mensajes publicados en `/application/in` se direccionan al destino `IN_QUEUE`, donde son seleccionados por la aplicación de la misma forma que cualquier otro mensaje colocado en esta cola por otras aplicaciones.

Obtención de mensajes de la aplicación

La aplicación está colocando mensajes en `OUT_QUEUE`, donde otros clientes los pueden seleccionar y procesar.

Sin embargo, en este caso, desea que los mensajes se entreguen a un cliente de AMQP en su lugar, pero los clientes de AMQP solo utilizan la publicación/suscripción y no pueden recoger mensajes directamente de una cola.

Para sustituir los clientes que previamente recibían mensajes con el cliente de AMP que realiza la suscripción, tendrá que crear un objeto de tema, para la serie de tema a la que está suscrito el cliente de AMQP, y una cola alias.

Atención: Si define la cola alias y, después, inicia la aplicación productora antes de que cualquier cliente de AMQP haya tenido la posibilidad de suscribirse, los mensajes que envía la aplicación productora a la "cola" (ahora un tema) se perderán porque no hay suscriptores.

Los cambios descritos en este texto sustituyen los clientes que previamente recibían los mensajes solo con el cliente de AMQP que realiza la suscripción. Para utilizar una combinación de AMQP y otros clientes para obtener mensajes, son necesarios cambios más amplios.

Dando por supuesto que el cliente de AMQP se suscribe a una serie de tema `/application/out`, puede definir el objeto de tema `APP_OUT` utilizando el mandato MQSC siguiente:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

Los mensajes entregados en este objeto de tema se entregan al cliente de AMQP que se suscribe a la misma serie de tema.

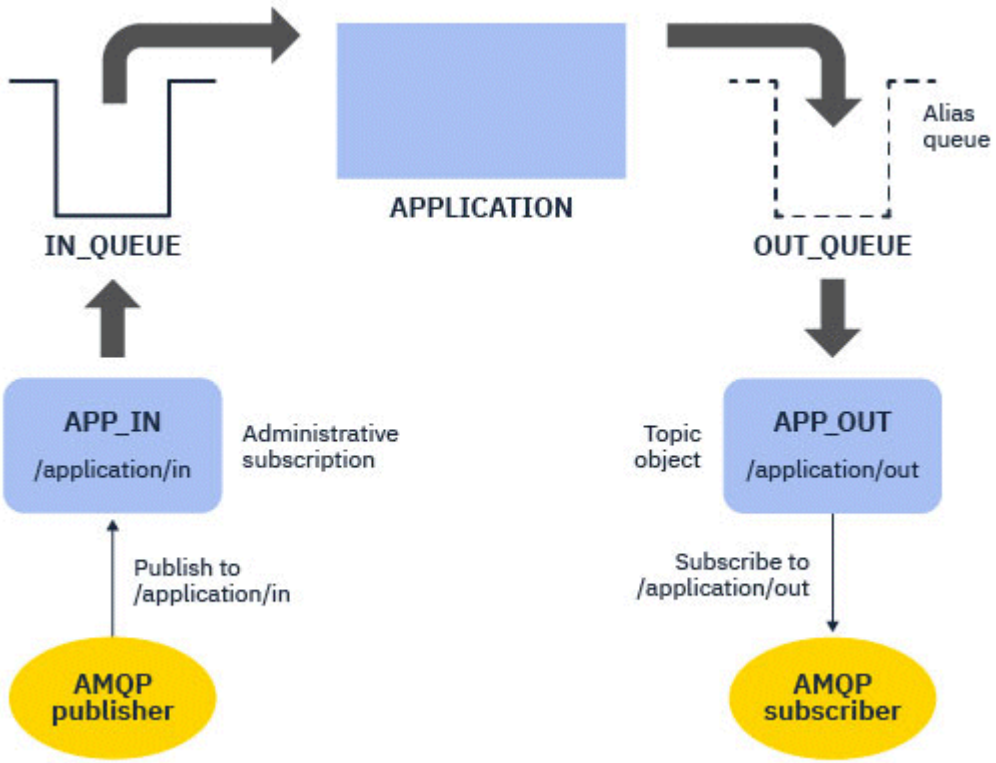
Tendrá que asegurarse de que los mensajes colocados en `OUT_QUEUE` por parte de la aplicación se entregan en este nuevo objeto de tema, de forma que se envían al cliente suscriptor.

Para ello, sustituya la cola existente `OUT_QUEUE` con una cola alias del mismo nombre, con un tipo de destino del objeto de tema que se acaba de crear, utilizando el mandato MQSC siguiente:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Ahora, los mensajes colocados por la aplicación en `OUT_QUEUE` no esperan en la cola a ser recogidos; en lugar de esto, se entregan en el destino de esta cola alias, es decir, el nuevo objeto de tema `APP_OUT`.

El cliente de AMQP, que está suscrito a la serie de tema representada por este objeto de tema `/application/out`, recibe los mensajes enviados a este objeto de tema desde la cola alias.



Tareas relacionadas

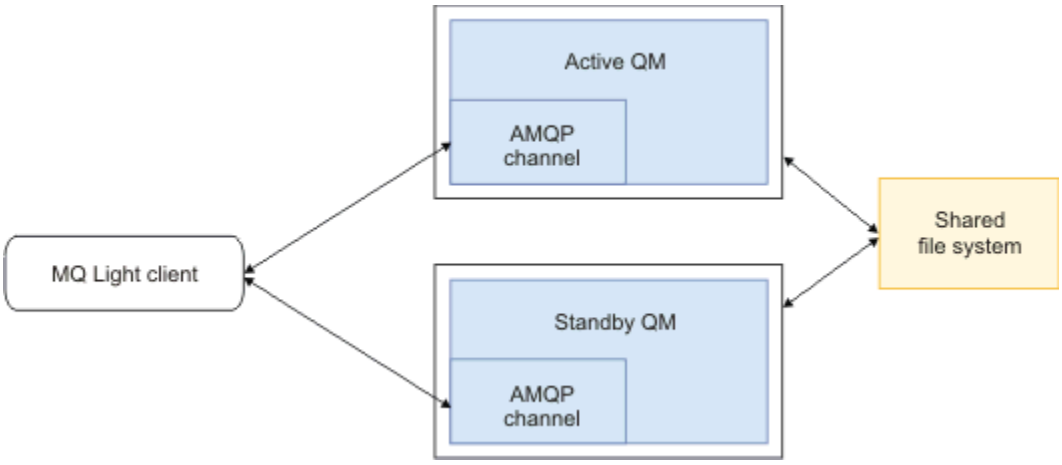
Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Configuración de un cliente de AMQP para la alta disponibilidad

Puede configurar aplicaciones AMQP 1.0 para conectarse a la instancia activa de un gestor de colas de varias instancias de IBM MQ y migrar tras error a la instancia en espera del gestor de colas de varias instancias en un par de alta disponibilidad (HA). Para ello, configure la aplicación AMQP con dos direcciones IP y pares de puerto.

Puede configurar la API de cliente AMQP con una función personalizada, a la que se llama si el cliente pierde su conexión con el servidor. La función puede conectarse a una dirección IP alternativa, por ejemplo, un gestor de colas IBM MQ en espera o la dirección IP original. Para otros clientes de AMQP, si el cliente da soporte a la configuración de varios puntos finales de conexión, configure la aplicación con dos pares de host-puerto y utilice las características de reconexión proporcionadas por la biblioteca de AMQP para conmutar al gestor de colas en espera.



Tareas relacionadas

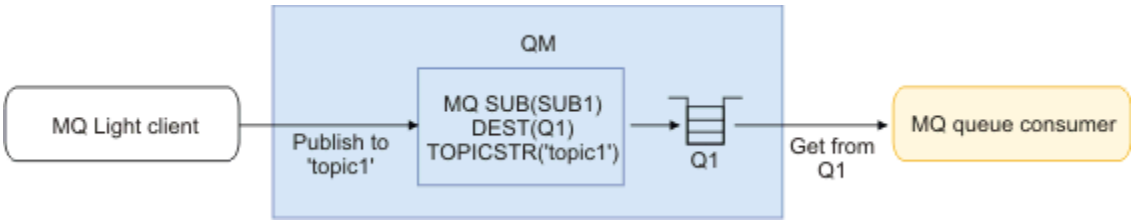
Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Configuración de la publicación/suscripción para clientes de AMQP

Los clientes de AMQP pueden publicar en un tema con una suscripción de IBM MQ que direcciona los mensajes a una cola IBM MQ que lee una aplicación existente. Si desea que una aplicación AMQP 1.0 envíe mensajes a una aplicación IBM MQ existente que esté configurada para leer de una cola, debe definir una suscripción de IBM MQ administrada en el gestor de colas.

Configure la suscripción para que utilice un patrón de tema que coincida con la serie de tema utilizada por la aplicación AMQP. Establezca el destino de la suscripción en el nombre de la cola que obtiene la aplicación IBM MQ o en la que examina mensajes.



Tareas relacionadas

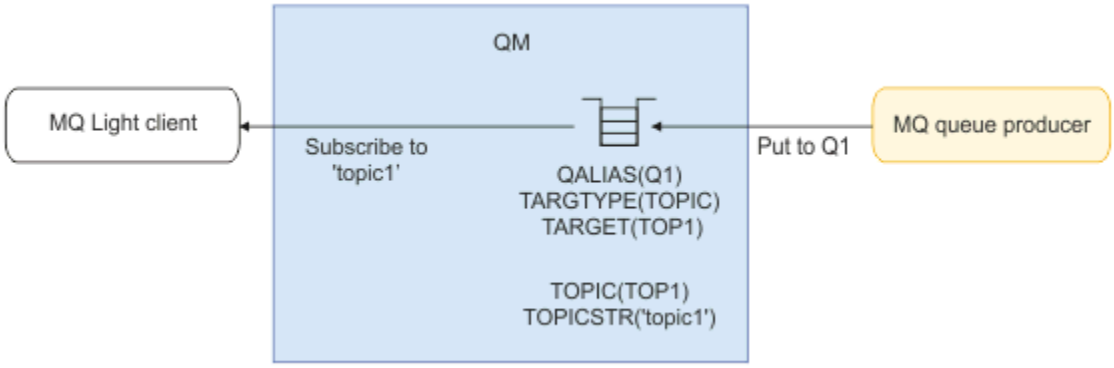
Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Cliente de AMQP que utiliza un alias de cola para recibir mensajes de una aplicación IBM MQ

Un cliente de AMQP puede suscribirse a un tema y recibir los mensajes colocados en una cola alias mediante una aplicación IBM MQ. Si desea que una aplicación AMQP 1.0 reciba mensajes de una aplicación IBM MQ existente que esté configurada para transferir mensajes a una cola, debe definir un alias de cola (QALIAS) en el gestor de colas.

El alias de cola debe tener el mismo nombre que la cola que abre la aplicación IBM MQ para la colocación. El alias de cola debe especificar un tipo base de TOPIC y un objeto base de un objeto de tema IBM MQ que tenga una serie de tema que coincida con el patrón de tema al que está suscrito mediante la aplicación AMQP.



Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Cliente de AMQP que envía solicitudes a y consume respuestas de un servidor de aplicaciones

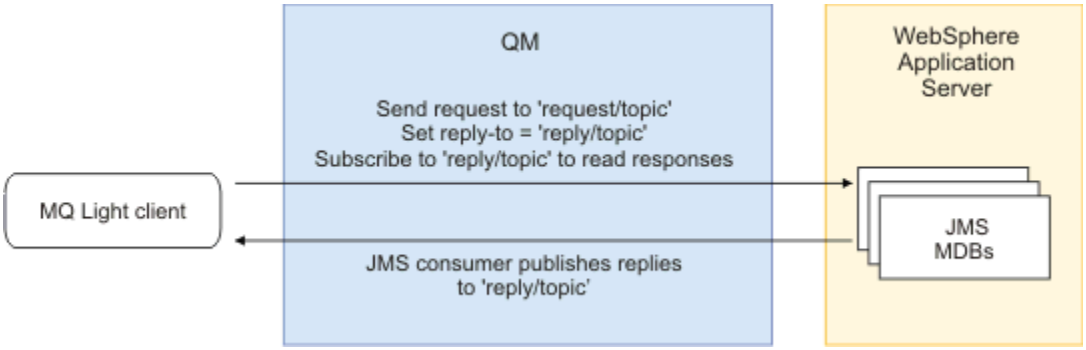
Un cliente AMQP puede enviar solicitudes a un bean controlado por mensajes que se ejecuta en un servidor de aplicaciones y consumir respuestas de un tema de respuesta. IBM MQ da soporte a las aplicaciones AMQP 1.0 estableciendo un tema de respuesta en los mensajes que publica IBM MQ. Cuando un mensaje AMQP se publica con el atributo de respuesta establecido, el valor del campo de respuesta se establece como una propiedad JMS para que la reciban los clientes JMS. Este valor permite a los clientes JMS leer el tema de respuesta del mensaje y enviar un mensaje de respuesta de vuelta al cliente de AMQP.

La propiedad JMS es **JMSReplyTo**. La serie de respuesta de AMQP debe ser de uno de los tipos siguientes:

- Una serie de tema. Por ejemplo: `'reply/topic'`
- Una dirección AMQP URL con el formato `amqp://host:port/[topic-string]`. Por ejemplo, `amqp://localhost:5672/reply/topic`

Si especifica un URL de dirección AMQP como el campo de respuesta, todo menos la serie de tema del final del URL se elimina antes de establecer la propiedad **JMSReplyTo**.

Para obtener más información sobre las correlaciones de una dirección de respuesta AMQP con una propiedad **JMSReplyTo** , consulte “Correlación entre campos de AMQP y campos de IBM MQ (mensajes entrantes)” en la página 705



Tareas relacionadas

Creación y utilización de canales AMQP

Protección de clientes de AMQP

ALW Interoperatividad entre aplicaciones MQ Light y Apache Qpid JMS

Las aplicaciones MQ Light y Apache Qpid JMS funcionan de formas similares y, al suscribirse a un tema, cree suscripciones de IBM MQ que sigan el mismo convenio de denominación.

Suscripción privada no compartida

El nombre de la suscripción de IBM MQ creada por la aplicación es `:private:<clientid>:<topicstring>`.

Una aplicación que utiliza un ID de cliente diferente no puede acceder a las suscripciones creadas por otras aplicaciones, porque el nombre de suscripción se genera automáticamente e incluye el ID de cliente AMQP.

Las aplicaciones Apache Qpid JMS y MQ Light utilizan este convenio de denominación para suscripciones privadas.

Suscripciones compartidas globalmente

El nombre de una suscripción IBM MQ compartida globalmente creada por un cliente AMQP es `:share:<sharename>:<topicstring>`.

Si varias aplicaciones con distintos ID de cliente AMQP especifican el mismo nombre de compartición y serie de tema, comparten una sola suscripción y pueden trabajar conjuntamente para procesar los mensajes para dicha suscripción. Puede utilizar este patrón si desea escalar el número de aplicaciones de trabajador que drenan mensajes de una suscripción.

Tanto las aplicaciones Apache Qpid JMS como MQ Light utilizan este convenio de denominación para suscripciones compartidas globalmente. En el caso de Apache Qpid JMS, esto requiere que la conexión JMS no tenga un ID de cliente especificado.

La biblioteca Apache Qpid JMS genera un ID de cliente AMQP automáticamente, pero este ID de cliente no se utiliza para la denominación de suscripciones de IBM MQ .

Nota: Las suscripciones compartidas globalmente siguen teniendo como ámbito un gestor de colas individual.

Suscripciones compartidas privadas

El nombre de una suscripción de IBM MQ compartida de forma privada creada por un cliente AMQP es `:privateshare:<clientid>:<sharename>:<topicstring>`.

Si varias hebras de una única aplicación Apache Qpid JMS utilizan el mismo nombre de compartición y serie de tema, y se ha configurado un ID de cliente en la conexión JMS , esas hebras comparten el mismo objeto de suscripción IBM MQ .

Sin embargo, otras conexiones Apache Qpid JMS no pueden compartir la suscripción porque deben utilizar un ID de cliente diferente.

Los clientes de MQ Light no soportan el concepto de suscripciones compartidas privadas y no pueden consumir mensajes de una suscripción compartida privada creada por una aplicación Apache Qpid JMS .

Suscripciones de IBM MQ JMS

Las suscripciones de IBM MQ JMS utilizan un esquema de denominación diferente de los canales AMQP. No es posible que las aplicaciones MQ Light o Apache Qpid JMS compartan suscripciones con aplicaciones IBM MQ JMS .

Conceptos relacionados

Desarrollo de aplicaciones cliente AMQP

El soporte de IBM MQ para las API de AMQP permite a un administrador de IBM MQ crear un canal AMQP. Cuando se inicia, este canal define un número de puerto que acepta conexiones procedentes de aplicaciones cliente AMQP.

ALW Propiedades de control de escucha AMQP de IBM MQ

Para obtener un mejor rendimiento en una aplicación multihebra, puede ajustar el número de hebras Worker que el servicio AMQP debe utilizar configurando una propiedad en el archivo de propiedades AMQP.

Puede configurar las propiedades del servicio de escucha AMQP en los siguientes archivos de propiedades:

- Windows En sistemas Windows : `amqp_win.properties` .
- Linux AIX En sistemas AIX and Linux : `amqp_unix.properties` .

Las propiedades que puede configurar son las siguientes:

Tabla 105. Propiedades del servicio de escucha AMQP	
Propiedad	**Descripción**
com.ibm.mq.MQXR.Workers	Número de hebras de trabajo de servidor que crea el servicio de escucha AMQP. Si no se especifica este valor, el valor predeterminado es igual al número de procesadores lógicos del sistema.
MQIBindType	El tipo de enlace para el servicio AMQP: FASTPATH, SHAREDo AISLAD0. El valor predeterminado es FASTPATH.

El servicio de escucha AMQP equilibra la carga de trabajo de conexión de cliente entre varias hebras de trabajo. El número de hebras de trabajo que el servicio AMQP debe utilizar se puede especificar utilizando la propiedad **com.ibm.mq.MQXR.Workers** .

El administrador del gestor de colas de IBM MQ puede ajustar el número de hebras de trabajo para obtener un mejor rendimiento en una aplicación multihebra. Normalmente, el mejor rendimiento se consigue cuando el número de hebras Worker coincide con el número de procesadores lógicos del sistema. Sin embargo, es posible que no siempre sea el caso para determinadas configuraciones de máquina y características de carga de cliente, por lo que es posible que sea necesario un elemento de ajuste para encontrar el valor óptimo para el número de hebras Worker.

Antes de realizar el ajuste, asegúrese de que comprende perfectamente la naturaleza de las aplicaciones cliente y sus cargas de trabajo. La medición del rendimiento de la aplicación con distintos recuentos de hebras y benchmarking debería ayudar a determinar el valor óptimo para el número de hebras de trabajo.

Nota: MQ Appliance Estas propiedades no son aplicables en IBM MQ Appliance. Los valores predeterminados se utilizan en IBM MQ Appliance.

Desarrollo de aplicaciones REST con IBM MQ

Se pueden desarrollar aplicaciones REST que envíen y reciban mensajes. IBM MQ soporta distintas API REST dependiendo de la plataforma y de la funcionalidad.

Las opciones siguientes son las opciones soportadas de IBM MQ entre las que puede elegir enviar mensajes a y recibir mensajes de, IBM MQ:

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

IBM MQ messaging REST API

Puede utilizar messaging REST API para enviar, recibir y examinar mensajes de IBM MQ en formato de texto sin formato. La messaging REST API está habilitada de forma predeterminada.

Se soportan una serie de cabeceras HTTP diferentes que se pueden utilizar para definir propiedades de mensaje comunes.

La messaging REST API está completamente integrada con la seguridad de IBM MQ. Para utilizar la messaging REST API, los usuarios deben autenticarse en el servidor mqweb y deben ser miembros del rol MQWebUser.

Para obtener más información, consulte “Mensajería utilizando la REST API” en la página 723. Consulte también Guía de aprendizaje: Cómo empezar con la IBM MQ REST API en IBM Developer, que incluye ejemplos de Go y Node.js para utilizar la API REST de mensajería.

IBM z/OS Conectar EE

IBM z/OS Connect EE es un producto de z/OS que le permite crear API REST sobre activos de z/OS existentes, como por ejemplo transacciones CICS o IMS y colas y temas de IBM MQ . El activo z/OS existente está oculto al usuario. Esto permite habilitar los activos para REST sin cambiar ninguno de ellos ni las aplicaciones existentes que los usan.

IBM z/OS Connect EE proporciona transformación automática de datos para convertir entre los datos JSON utilizados por las API REST y las estructuras de lenguaje más tradicionales, por ejemplo COBOL, que esperan muchas aplicaciones de sistema principal.

El kit de herramientas de API de IBM z/OS Connect EE basado en Eclipse se puede utilizar para crear una API RESTful completa utilizando parámetros de consulta y segmentos de vía de acceso URL , manipulando el formato JSON a medida que fluye a través del tiempo de ejecución de IBM z/OS Connect EE.

IBM z/OS Connect EE se puede utilizar para exponer colas y temas de IBM MQ como API RESTful a través del proveedor de servicios IBM MQ . Se soportan dos tipos de servicio:

- Servicios unidireccionales: proporcionan una API REST que permite realizar una única operación de IBM MQ en una cola o tema. En función de la configuración exacta, una solicitud HTTP puede hacer que un mensaje se envíe a una cola o se publique en un tema; o una solicitud HTTP puede hacer que un mensaje se reciba de forma destructiva de una cola
- Servicios bidireccional: proporcionan una API REST encima de un par de colas utilizadas por una aplicación de estilo de solicitud-respuesta de fondo. Los llamantes emiten una solicitud HTTP al servicio bidireccional. La carga útil de solicitud HTTP se transforma de JSON a una estructura de lenguaje tradicional y se coloca en una cola de solicitudes donde la procesa la aplicación de fondo y una respuesta se coloca en la cola de respuestas. El servicio recupera esta respuesta, la convierte de la estructura de lenguaje tradicional a JSON y la devuelve al llamante como cuerpo de respuesta POST.

Para obtener más información sobre IBM z/OS Connect EE, consulte z/OS Connect EE.

Para obtener más información sobre el proveedor de servicios de IBM MQ, consulte Utilización del proveedor de servicios de IBM MQ.

IBM Integration Bus

IBM Integration Bus es la IBMque se puede utilizar para conectar aplicaciones y sistemas independientemente de los formatos de mensaje y protocolos que soporten.

IBM Integration Bus siempre ha soportado IBM MQ y proporciona nodos *HTTPInput* y *HTTPRequest*, que se pueden usar para construir una interfaz RESTful encima de IBM MQ y muchos otros sistemas como, por ejemplo, bases de datos.

IBM Integration Bus se puede utilizar para mucho más que proporcionar una simple interfaz REST encima de IBM MQ. Sus prestaciones pueden usarse para proporcionar una manipulación avanzada de las cargas útiles y muchas otras mejoras como parte de un REST API.

Para obtener información adicional, consulte el ejemplo de tecnología, que expone una interfaz JSON sobre REST encima de una aplicación IBM MQ que espera una carga útil en XML.

DataPower

La pasarela DataPower es una única pasarela multicanal que ayuda a proporcionar seguridad, control, integración y un acceso optimizado a una gama de sistemas, incluyendo IBM MQ. Se presenta en factores de forma tanto hardware como virtuales.

Uno de los servicios que proporciona DataPower es una pasarela multiprotocolo que puede tomar la entrada en un protocolo y generar la salida en un protocolo distinto. En concreto, DataPower se puede configurar para aceptar datos HTTP(S) y direccionarlo a IBM MQ a través de una conexión cliente, que se pueden utilizar para crear una interfaz REST encima de IBM MQ. Otros servicios de DataPower como, por ejemplo, la transformación, también se pueden usar para mejorar la interfaz REST.

Para obtener más información, consulte Pasarela multiprotocolo.

Mensajería utilizando la REST API

Puede utilizar messaging REST API para realizar una mensajería simple punto a punto y de publicación. Puede publicar mensajes en un tema, enviar mensajes a una cola, examinar mensajes en una cola y obtener de forma destructiva mensajes de una cola. La información se envía y recibe de la messaging REST API como texto sin formato.

Antes de empezar

Nota:

- La messaging REST API está habilitada de forma predeterminada. Puede inhabilitar la messaging REST API para evitar toda la mensajería. Para obtener más información sobre la habilitación o la inhabilitación de la messaging REST API, consulte Configuración de la messaging REST API.
- La messaging REST API se integra con la seguridad de IBM MQ. Para utilizar la messaging REST API, los usuarios deben autenticarse en el servidor mqweb y deben ser miembros del rol MQWebUser. El usuario también debe tener autorización para acceder a la cola o tema especificado. Para obtener más información sobre la seguridad para REST API, consulte Seguridad de IBM MQ Console y REST API.
- Si utiliza Advanced Message Security (AMS) con la messaging REST API, tenga en cuenta que todos los mensajes se cifran utilizando el contexto del servidor mqweb, no el contexto del usuario que publica el mensaje.
- Al recibir o examinar un mensaje, solo se da soporte a los mensajes con formato IBM MQ `MQSTR` o JMS `TextMessage` . Posteriormente, todos los mensajes se reciben de forma destructiva bajo el punto de sincronización y los mensajes no manejados se dejan en la cola. La cola de IBM MQ puede configurarse para trasladar estos mensajes con formato incorrecto a un destino alternativo. Para obtener más información, consulte “Manejo de mensajes no entregables en IBM MQ classes for JMS” en la página 240.
- La messaging REST API no proporciona una entrega de una vez (y solo una) de mensajes con soporte transaccional. Si se emite un POST HTTP y la conexión falla antes de que el cliente reciba una respuesta HTTP , el cliente no puede indicar inmediatamente si el mensaje se ha enviado a la cola especificada o se ha publicado en el tema especificado. Si se emite un HTTP DELETE y la conexión falla antes de que el cliente reciba una respuesta HTTP, se podría haber obtenido un mensaje de forma destructiva y perderse, ya que no hay ninguna forma de recuperarse de la destrucción.
- V 9.3.0 A partir de IBM MQ 9.3.0, las líneas nuevas de las series entrantes ya no se eliminan mediante la operación POST HTTP . Las aplicaciones REST que utilizan versiones anteriores no deben utilizar líneas nuevas en los mensajes que se envían o publican utilizando la API REST, ya que se perderán.

Procedimiento

- “Iniciación a messaging REST API” en la página 724
- “Utilización de messaging REST API” en la página 726
- REST API tratamiento de errores
- Descubrimiento de REST API
- Soporte multilingüístico de REST API

Referencia relacionada

Referencia de la REST API de mensajería

Información relacionada

Guía de aprendizaje: Cómo empezar con la mensajería de IBM MQ REST API

Iniciación a messaging REST API

Iníciese rápidamente en la messaging REST API y pruebe unos cuantos mandatos de ejemplo utilizando cURL.

Antes de empezar

Para empezar a utilizar la messaging REST API, los ejemplos de esta tarea tienen los requisitos siguientes:

- Los ejemplos utilizan cURL para enviar solicitudes REST para poner y obtener mensajes de una cola. Por lo tanto, para completar estar, necesita tener el cURL instalado en el sistema.
- Los ejemplos utilizan un gestor de colas `QM1`. Cree un gestor de colas con el mismo nombre o sustituya un gestor de colas existentes en el sistema. El gestor de colas debe estar en la misma máquina que el servidor mqweb.
- Para completar esta tarea, debe ser un usuario con determinados privilegios para que pueda utilizar el mandato **dspmqweb**:
  - z/OS En z/OS, debe tener autorización para ejecutar el mandato **dspmqweb** y acceso de escritura en el archivo `mqwebuser.xml`.
  - Multi En todos los demás sistemas operativos, debe ser un usuario con privilegios.

IBM i En IBM i, los mandatos deben estar en ejecución en QSHELL.

Procedimiento

1. Asegúrese de que el servidor mqweb está configurado para la messaging REST API:
   - Asegúrese de que ha configurado el servidor mqweb para que lo utilice administrative REST API, administrative REST API para MFT, messaging REST APIo IBM MQ Console.Para obtener más información sobre cómo configurar el servidor mqweb con un registro básico, consulte Configuración básica para el servidor mqweb.
   - Si el servidor mqweb ya está configurado, asegúrese de que ha añadido los usuarios adecuados para habilitar la mensajería en el paso 5 de Configuración básica para el servidor mqweb.
     - Si **mqRestMessagingAdoptWebUserContext** se establece en `true` en la configuración del servidor mqweb, los usuarios de messaging REST API deben ser miembros del rol `MQWebUser` . Los roles `MQWebAdmin` y `MQWebAdminRO` no son aplicables para messaging REST API. Los usuarios también deben tener autorización para acceder a colas y temas que se utilizan para la mensajería a través de OAM o RACF.
     - Si **mqRestMessagingAdoptWebUserContext** se establece en `false` en la configuración del servidor mqweb, el ID de usuario que se utiliza para iniciar el servidor mqweb debe tener autorización para acceder a las colas utilizadas para la mensajería a través de OAM o RACF.
2. z/OS
   En z/OS, establezca la variable de entorno WLP_USER_DIR para que pueda utilizar el mandato **dspmqweb** . Establezca la variable para que apunte a la configuración del servidor mqweb especificando el mandato siguiente.

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   donde *WLP_user_directory* es el nombre del directorio que se pasa a `crtmqweb`. Por ejemplo:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   Si desea más información, consulte Creación del servidor mqweb.
3. Determine el URL de REST API especificando el mandato siguiente:

   ```
   dspmqweb status
   ```

Los ejemplos de los pasos siguientes presuponen que el REST API URL de URL es el URL `https://localhost:9443/ibmmq/rest/v2/`predeterminado. Si el URL es diferente al valor predeterminado, sustituya el URL en los pasos siguientes.

4. Cree una cola, `MSGQ`, en el gestor de colas `QM1`. Esta cola se utiliza para la mensajería. Utilice uno de los métodos siguientes:

   - Utilice una solicitud POST en el recurso `mqsc` de administrative REST API, autenticándose como el usuario `mqadmin` :

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/
     json" --data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
     \"qlocal\",\"name\": \"MSGQ\"}"
     ```

   - Utilice mandatos MQSC:

     z/OS En z/OS, utilice un origen 2CR en lugar del mandato **runmqsc**. Para obtener más información, consulte Orígenes desde los que puede emitir mandatos MQSC y PCF en IBM MQ for z/OS.

     a. Inicie **runmqsc** para el gestor de colas especificando el mandato siguiente:

        ```
        runmqsc QM1
        ```

     b. Utilice el mandato MQSC **DEFINE QLOCAL** para crear la cola:

        ```
        DEFINE QLOCAL(MSGQ)
        ```

     c. Salga del mandato **runmqsc** especificando el mandato siguiente:

        ```
        end
        ```

5. Otorgue autorización al usuario que ha añadido a `mqwebuser.xml` en el paso 5 de Configuración básica para el servidor mqweb para acceder a la cola `MSGQ`. Sustituya el usuario donde se utilice `myuser`:

   - z/OS En z/OS:

     a. Otorgue al usuario acceso a la cola:

        ```
        RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
        PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
        ```

     b. Otorgue al ID de usuario de la tarea iniciada por mqweb acceso para establecer todo el contexto de la cola:

        ```
        RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
        PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
        ```

   - Multi En todos los demás sistemas operativos, si el usuario está en el grupo mqm, ya se ha otorgado la autorización. De lo contrario, especifique los mandatos siguientes:

     a. Inicie **runmqsc** para el gestor de colas especificando el mandato siguiente:

        ```
        runmqsc QM1
        ```

     b. Utilice el mandato MQSC **SET AUTHREC** para otorgar al usuario autorización para examinar, consultar, obtener y poner en la cola:

        ```
        SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
        PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
        ```

c. Salga del mandato **runmqsc** especificando el mandato siguiente:

```
end
```

6. Coloque un mensaje con el contenido `Hello World!` en la cola `MSGQ` en el gestor de colas `QM1`, utilizando una solicitud POST en el recurso `message` . Sustituya el ID de usuario y contraseña de `mqwebuser.xml` por `myuser` y `mypassword`:

   Se utiliza la autenticación básica y una cabecera HTTP `ibm-mq-rest-csrf-token` con un valor arbitrario se establece en la solicitud REST cURL. Esta cabecera adicional es necesaria para las solicitudes POST, PATCH y DELETE.

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X
   POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
   plain;charset=utf-8" --data "Hello World!"
   ```

7. Obtenga de forma destructiva el mensaje de la cola `Hello World!` en la cola `MSGQ` en el gestor de colas `QM1`, utilizando una solicitud DELETE en el recurso `message` . Sustituya el ID de usuario y contraseña de `mqwebuser.xml` por `myuser` y `mypassword`:

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
   ```

   Se devuelve el mensaje `Hello World!` .

Qué hacer a continuación

- Los ejemplos utilizan la autenticación básica para proteger la solicitud. En su lugar, puede utilizar la autenticación basada en señal o la autenticación basada en cliente. Si desea más información, consulte Utilización de autenticación de certificado de cliente con la REST API y IBM MQ Console, y Utilización de la autenticación basada en señal con la REST API.
- Obtenga más información sobre cómo utilizar messaging REST API y construir los URL con parámetros de consulta: “Utilización de messaging REST API” en la página 726.
- Cuando se utiliza messaging REST API, las conexiones con el gestor de colas se agrupan para optimizar el rendimiento. Puede configurar el tamaño máximo de la agrupación, y la acción que se realiza cuando se están utilizando todas las conexiones de la agrupación: Configuración de messaging REST API.
- Examine la información de referencia para los recursos de messaging REST API disponibles y todos los parámetros de consulta disponibles: Referencia de messaging REST API.
- Descubra la administrative REST API, una interfaz RESTful para la administración de IBM MQ: Administración utilizando la REST API.
- Descubra la IBM MQ Console, una GUI basada en navegador: Administración utilizando la IBM MQ Console.

Utilización de messaging REST API

Cuando utiliza la messaging REST API, invoca métodos HTTP en un URL para enviar y recibir mensajes de IBM MQ. El método HTTP, por ejemplo, POST representa el tipo de acción que debe realizarse en el objeto que está representado por el URL. Puede existir más información sobre la acción, codificada dentro de parámetros de consulta. La información sobre el resultado de realizar la acción puede devolverse como el cuerpo de la respuesta HTTP.

Antes de empezar

Tenga en cuenta estos puntos antes de utilizar messaging REST API:

- Debe autenticarse con el servidor mqweb para poder utilizar messaging REST API. Puede autenticarse mediante la autenticación básica HTTP, la autenticación de certificados de cliente o la autenticación basada en señal. Para obtener más información sobre cómo utilizar estos métodos de autenticación, consulte Seguridad deIBM MQ Console y REST API.

- REST API es sensible a las mayúsculas y minúsculas. Por ejemplo, una acción HTTP POST en el URL siguiente produce un error si el gestor de colas se denomina `qmgr1`.

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- V 9.3.3 Si se está conectando a un gestor de colas remoto con messaging REST API, debe utilizar el nombre exclusivo para la conexión del gestor de colas en lugar del nombre del gestor de colas.
- No todos los caracteres que se pueden utilizar en nombres de objeto de IBM MQ se pueden codificar directamente en un URL. Para codificar estos caracteres correctamente, debe utilizar la codificación de URL apropiada:
  - Una barra inclinada se debe codificar como %2F.
  - Un signo de porcentaje se debe codificar como %25.
  - Un punto se debe codificar como %2E.
  - Un signo de interrogación se debe codificar como %3F.
- Al recibir o examinar un mensaje, sólo se da soporte a los mensajes con formato IBM MQ `MQSTR` y JMS `TextMessage` . Posteriormente, todos los mensajes se reciben de forma destructiva bajo el punto de sincronización y los mensajes no manejados se dejan en la cola. La cola de IBM MQ puede configurarse para trasladar estos mensajes con formato incorrecto a un destino alternativo. Para obtener más información, consulte “Manejo de mensajes no entregables en IBM MQ classes for JMS” en la página 240.

Acerca de esta tarea

Cuando utiliza la REST API para realizar una acción de mensajería en un objeto de cola de IBM MQ, primero necesita construir un URL para representar ese objeto. Cada URL empieza con un prefijo, que describe a qué nombre de host y puerto se debe enviar la solicitud. El resto del URL describe un objeto determinado, o una ruta a ese objeto, conocido como recurso.

La acción de mensajería que se debe realizar en el recurso define si el URL necesita parámetros de consulta o no. También define el método HTTP que se utiliza, y si se envía información adicional al URL o se devuelve de ella. La información adicional puede formar parte de la solicitud HTTP o devolverse como parte de la respuesta HTTP.

Después de construir el URL, puede enviar la solicitud HTTP a IBM MQ. Puede enviar la solicitud utilizando la implementación de HTTP que se base en el lenguaje de programación de su elección. También puede enviar la solicitud utilizando herramientas de línea de mandatos, tales como cURL, o un navegador web o un complemento de navegador web.

Importante: Como mínimo, hay que seguir los pasos “1.a” en la página 727 y “1.b” en la página 727.

Procedimiento

1. Construya el URL:
   a) Determine el URL del prefijo especificando el mandato siguiente:

      ```
      dspmqweb status
      ```

      El URL que desea utilizar incluye la frase `/ibmmq/rest/`.

   b) Añada la cola y el gestor de colas asociado que se deben utilizar para la mensajería en la vía de acceso de URL.

      En la referencia de mensajería, los segmentos de variable se pueden identificar en el URL mediante las llaves que lo rodean *{ }*. Para obtener más información, consulte /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

Por ejemplo, para interactuar con la cola *Q1* asociada al gestor de colas *QM1*, añada `/qmgr` y `/queue` al URL de prefijo para crear el URL siguiente:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
```

Consejo: V 9.3.3 Si el gestor de colas es un gestor de colas remoto, debe utilizar el nombre exclusivo para el gestor de colas en lugar del nombre del gestor de colas. El gestor de colas remoto debe estar configurado antes de que se pueda utilizar con el messaging REST API. Para obtener más información, consulte "Configuración de un gestor de colas remoto para utilizarlo con messaging REST API" en la página 728.

c) Opcional: Añada un parámetro de consulta opcional al URL.

Añada un signo de interrogación,?, parámetro de consulta, signo igual = y un valor para el URL.

Por ejemplo, para esperar un máximo de 30 segundos para que el mensaje siguiente esté disponible, cree el URL siguiente:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) Opcional: Añada parámetros de consulta opcionales adicionales al URL.

Añada un ampersand, &, al URLy, a continuación, repita el paso 1c.

2. Invoque el método HTTP correspondiente en el URL. Especifique cualquier carga útil de mensaje opcional y proporcione las credenciales de seguridad adecuadas para la autenticación. Por ejemplo:
   - Utilice la implementación de HTTP/REST del lenguaje de programación elegido.
   - Utilice una herramienta tal como el complemento de navegador de cliente REST o cURL.

V 9.3.3 Configuración de un gestor de colas remoto para utilizarlo con messaging REST API

Puede utilizar messaging REST API para conectarse a gestores de colas remotos para la mensajería. Para poder conectarse a un gestor de colas remoto, debe configurar la configuración del gestor de colas remoto. A continuación, puede conectarse al gestor de colas remoto utilizando el nombre exclusivo definido en la información de configuración.

Antes de empezar

- Asegúrese de que ha configurado el servidor mqweb para que lo utilice administrative REST API, administrative REST API para MFT, messaging REST APIo IBM MQ Console.Para obtener más información sobre cómo configurar el servidor mqweb con un registro básico, consulte Configuración básica para el servidor mqweb.
- Si el servidor mqweb ya está configurado, asegúrese de que ha añadido los usuarios adecuados para habilitar la mensajería en el paso 5 de Configuración básica para el servidor mqweb. Los usuarios de messaging REST API deben ser miembros del rol `MQWebUser` . Los roles `MQWebAdmin` y `MQWebAdminRO` no son aplicables para messaging REST API.
  - Si **`mqRestMessagingAdoptWebUserContext`** se establece en `true` en la configuración del servidor mqweb, los usuarios con el rol `MQWebUser` deben tener autorización para acceder a colas y temas que se utilizan para la mensajería a través de OAM o RACF.
  - Si **`mqRestMessagingAdoptWebUserContext`** se establece en `false` en la configuración del servidor mqweb, el ID de usuario que se utiliza para iniciar el servidor mqweb debe tener autorización para acceder a colas y temas que se utilizan para la mensajería a través de OAM o RACF.
- Asegúrese de que el messaging REST API esté configurado para conectarse a gestores de colas remotos. Para obtener más información, consulte Configuración de la modalidad de conexión para messaging REST API.

Acerca de esta tarea

Puede conectarse a gestores de colas remotos utilizando la messaging REST API. Un gestor de colas remoto puede ser un gestor de colas en otro sistema, un gestor de colas en otra instalación o un gestor de colas en la misma instalación que el servidor mqweb.

Para conectarse a un gestor de colas remoto, debe completar los pasos de configuración siguientes:

- Configure un canal de conexión de servidor y un escucha.
- Otorgue autorización a un usuario adecuado para acceder al gestor de colas.
- Cree un archivo CCDT que contenga la información de conexión para el gestor de colas.
- Añada la información de conexión a messaging REST API utilizando el mandato **setmqweb remote** .

A continuación, puede utilizar el gestor de colas remoto proporcionando el nombre exclusivo en el recurso URL en lugar del nombre del gestor de colas.

También puede configurar los gestores de colas remotos como parte de un grupo de gestores de colas. Para obtener más información, consulte "Configuración de un grupo de gestores de colas para utilizarlo con la API REST de mensajería" en la página 731.

Procedimiento

1. En el gestor de colas remoto, cree un canal de conexión de servidor para permitir conexiones remotas con el gestor de colas. Puede crear canales de conexión con el servidor utilizando el mandato MQSC de **DEFINE CHANNEL** en la línea de mandatos.
   Por ejemplo, para crear un canal de conexión de servidor QM1.SVRCONN para el gestor de colas QM1, especifique el mandato siguiente:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Para obtener más información sobre **DEFINE CHANNEL** y las opciones disponibles, consulte DEFINE CHANNEL.
2. Asegúrese de que un usuario adecuado esté autorizado para acceder al gestor de colas. Este usuario también debe tener autorización para acceder a cualquier cola o tema que utilice para la mensajería. El usuario necesita autorización `connect`, `inquire`, `alternate usery set context` sobre el gestor de colas. En UNIX, Linux, and Windows , utilice el mandato de control **setmqaut** en la línea de mandatos. En z/OS, defina perfiles RACF para otorgar al usuario autorizado acceso al gestor de colas. Por ejemplo, en UNIX, Linux, and Windows, para autorizar a un usuario, `exampleUser`, a acceder al gestor de colas QM1, especifique el mandato siguiente:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Para obtener más información sobre qué usuario debe estar autorizado, consulte "Determinación del principal de seguridad utilizado por messaging REST API" en la página 734.
3.
   Si no existe ningún escucha en el gestor de colas remoto, cree un escucha para aceptar conexiones de red entrantes utilizando el mandato MQSC de **DEFINE LISTENER** en la línea de mandatos.
   Por ejemplo, para crear un escucha REMOTE.LISTENER en el puerto 1414 para el gestor de colas remoto QM1, especifique el mandato siguiente:

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. Asegúrese de que el escucha se está ejecutando utilizando el mandato MQSC de **START LISTENER** en la línea de mandatos:

ALW Por ejemplo, en AIX, Linux, and Windows para iniciar el escucha REMOTE.LISTENER para el gestor de colas QM1, especifique el mandato siguiente:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS Por ejemplo, en z/OS, para iniciar el escucha, entre el mandato siguiente:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

El espacio de direcciones del iniciador de canal debe iniciarse antes de poder iniciar un escucha en z/OS.

5. En el sistema en el que se ejecuta el servidor mqweb que aloja el messaging REST API , cree o actualice un archivo CCDT JSON que contenga la información de conexión del gestor de colas.

   El archivo CCDT debe incluir la información de `name`, `clientConnection` y `type` . Opcionalmente, puede incluir información adicional como, por ejemplo, la información de `transmissionSecurity`. Para obtener más información sobre todas las definiciones de atributos de canal de CCDT, consulte la Lista completa de definiciones de atributos de canal CCDT.

   El ejemplo siguiente muestra un archivo CCDT JSON básico para una conexión de gestor de colas remoto. Establece el nombre del canal en el mismo nombre que el canal de conexión de servidor de ejemplo creado en el paso “1” en la página 729. El puerto de conexión se establece en el mismo valor que el puerto utilizado por el escucha. El host de conexión se establece en el nombre de host del sistema en el que se ejecuta el gestor de colas remoto, QM1.

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "example.com",
               "port": 1414
             }],
           "queueManager": "QM1"
         },
         "type": "clientConnection"
       }]
   }
   ```

6. En la instalación que ejecuta el servidor mqweb que aloja el messaging REST API, utilice el mandato **setmqweb remote** para añadir la información del gestor de colas remoto a la configuración del servidor mqweb.

   Como mínimo, debe especificar los parámetros siguientes:

   - **-qmgrName**, donde se especifica el nombre del gestor de colas.
   - **-ccdtURL**, donde especifica el URL de CCDT para el gestor de colas.
   - **-uniqueName**, donde se especifica un nombre exclusivo para el gestor de colas. El nombre exclusivo se utiliza para diferenciar los gestores de colas remotos que pueden tener el mismo nombre y, por lo tanto, no debe existir para identificar otro gestor de colas.

   Puede especificar otras opciones, como el nombre de usuario y la contraseña que se van a utilizar para la conexión del gestor de colas remoto, o los detalles del almacén de confianza y del almacén de claves. Para obtener una lista completa de los parámetros que se pueden especificar con el mandato **setmqweb remote** , consulte setmqweb remote.

   Por ejemplo, para añadir el gestor de colas remoto QM1, con el archivo CCDT de ejemplo, especifique el mandato siguiente:

   ```
   setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
   ```

Resultados

El gestor de colas remoto se puede utilizar con messaging REST API utilizando el nombre exclusivo en el recurso URL en lugar del nombre del gestor de colas.

Ejemplo

El ejemplo siguiente configura la conexión del gestor de colas remoto para un gestor de colas QM1. El IBM MQ Console autorizado para administrar el gestor de colas basándose en la autorización que se otorga al usuario exampleUser. Las credenciales de este usuario se proporcionan a IBM MQ Console cuando se utiliza **setmqweb remote** para configurar la conexión del gestor de colas.

1. En el sistema donde está el gestor de colas remoto QM1 , se crean un canal de conexión de servidor y un escucha. El escucha se inicia y se otorga autorización al usuario `exampleUser` para conectarse al gestor de colas y acceder a una cola que se utiliza para la mensajería:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. En el sistema donde se ejecuta el servidor mqweb, se crea un archivo `QM1_ccdt.json` con la siguiente información de conexión:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. En el sistema donde se ejecuta el servidor mqweb, la información de conexión para el gestor de colas QM1 se añade al servidor mqweb. Las credenciales para `exampleUser` se incluyen en la información de conexión:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. El messaging REST API puede conectarse al gestor de colas remoto QM1 utilizando el nombre exclusivo para la conexión del gestor de colas en lugar del nombre del gestor de colas en el recurso URL:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.3.3 Configuración de un grupo de gestores de colas para utilizarlo con la API REST de mensajería

Puede utilizar messaging REST API para conectarse a grupos de gestores de colas para la mensajería. Para poder conectarse a un grupo de gestores de colas, debe configurar la configuración del gestor de

colas remoto para el grupo. A continuación, puede conectarse al grupo de gestores de colas utilizando el nombre exclusivo definido en la información de configuración.

Antes de empezar

- Asegúrese de que ha configurado el servidor mqweb para que lo utilice administrative REST API, administrative REST API para MFT, messaging REST APIo IBM MQ Console.Para obtener más información sobre cómo configurar el servidor mqweb con un registro básico, consulte Configuración básica para el servidor mqweb.
- Si el servidor mqweb ya está configurado, asegúrese de que ha añadido los usuarios adecuados para habilitar la mensajería en el paso 5 de Configuración básica para el servidor mqweb. Los usuarios de messaging REST API deben ser miembros del rol `MQWebUser` . Los roles `MQWebAdmin` y `MQWebAdminRO` no son aplicables para messaging REST API.
  - Si **`mqRestMessagingAdoptWebUserContext`** se establece en `true` en la configuración del servidor mqweb, los usuarios con el rol `MQWebUser` deben tener autorización para acceder a las colas y temas que se utilizan para la mensajería. Puede autorizar a estos usuarios a través de OAM o RACF.
  - Si **`mqRestMessagingAdoptWebUserContext`** se establece en `false` en la configuración del servidor mqweb, el ID de usuario que inicia el servidor mqweb debe tener autorización para acceder a las colas y temas que se utilizan para la mensajería. Puede autorizar a este usuario a través de OAM o RACF.
- Asegúrese de que el messaging REST API esté configurado para conectarse a gestores de colas remotos. Para obtener más información, consulte Configuración de la modalidad de conexión para messaging REST API

Acerca de esta tarea

Un grupo de gestores de colas le permite conectar aplicaciones a cualquier gestor de colas dentro del grupo. El grupo se define como un conjunto de conexiones en una tabla de definición de canal de cliente (CCDT). Cuando se utiliza una llamada MQCONN o MQCONNX, se hace referencia al grupo añadiendo un asterisco como prefijo al nombre del gestor de colas. Con messaging REST API, puede hacer referencia al grupo utilizando el nombre exclusivo que está asociado con el grupo de gestores de colas. El nombre exclusivo se incluye en el recurso URL en lugar del nombre del gestor de colas. Para obtener más información sobre los grupos de gestores de colas, consulte “Grupos de gestores de colas en la CCDT” en la página 947.

También puede configurar los gestores de colas remotos individualmente. Para obtener más información, consulte “Configuración de un gestor de colas remoto para utilizarlo con messaging REST API” en la página 728.

Procedimiento

1. En cada uno de los gestores de colas remotos del grupo, cree un canal de conexión de servidor para permitir conexiones remotas con el gestor de colas. Puede utilizar el mandato MQSC de **`DEFINE CHANNEL`** en la línea de mandatos para crear canales de conexión de servidor.
   Por ejemplo, para crear un canal de conexión de servidor `QM1.SVRCONN` para el gestor de colas QM1, especifique el mandato siguiente:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Para obtener más información sobre **`DEFINE CHANNEL`** y las opciones disponibles, consulte DEFINE CHANNEL.
2. En cada uno de los gestores de colas remotos del grupo, asegúrese de que un usuario adecuado tenga autorización para acceder al gestor de colas. Este usuario también debe tener autorización para acceder a cualquier cola o tema que utilice para la mensajería. El usuario necesita autorización `connect`, `inquire`, `alternate usery set context` sobre el gestor de colas. En UNIX, Linux, and

Windows , utilice el mandato de control **setmqaut** en la línea de mandatos. En z/OS, defina perfiles RACF para otorgar al usuario autorizado acceso al gestor de colas.
Por ejemplo, en UNIX, Linux, and Windows, especifique el mandato siguiente para autorizar a un usuario, `exampleUser`, a acceder al gestor de colas QM1:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

Para obtener más información sobre qué usuario debe estar autorizado, consulte "Determinación del principal de seguridad utilizado por messaging REST API" en la página 734.

3. ALW
Si no existe ningún escucha en cada uno de los gestores de colas remotos del grupo, cree escuchas para aceptar conexiones de red entrantes. Puede utilizar el mandato MQSC de **DEFINE LISTENER** en la línea de mandatos para crear escuchas.
Por ejemplo, para crear un escucha `REMOTE.LISTENER` en el puerto 1414 para el gestor de colas remoto QM1, especifique el mandato siguiente:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. En cada uno de los gestores de colas remotos del grupo, asegúrese de que el escucha se esté ejecutando utilizando el mandato MQSC de **START LISTENER** en la línea de mandatos.

ALW Por ejemplo, en AIX, Linux, and Windows para iniciar el escucha `REMOTE.LISTENER` para el gestor de colas QM1, especifique el mandato siguiente:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS Por ejemplo, en z/OS, para iniciar el escucha, entre el mandato siguiente:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

El espacio de direcciones del iniciador de canal debe iniciarse antes de poder iniciar un escucha en z/OS.

5. En el sistema donde se ejecuta el servidor mqweb que aloja el messaging REST API , cree un archivo CCDT JSON. Este archivo JSON contiene información de conexión para cada gestor de colas del grupo.

El archivo CCDT debe incluir la información de `name`, `clientConnection`y `type` para cada conexión de gestor de colas. Opcionalmente, puede incluir información adicional como, por ejemplo, la información de `transmissionSecurity`. Para obtener más información sobre todas las definiciones de atributos de canal de CCDT, consulte la Lista completa de definiciones de atributos de canal CCDT.

El ejemplo siguiente muestra un archivo CCDT JSON básico para dos conexiones de gestor de colas. La primera conexión es para el gestor de colas QM1. Tiene un canal de conexión de servidor de `QM1.SVRCONN`, un escucha en el puerto 1414y se ejecuta en el host `QM1.example.com`. La segunda conexión es para el gestor de colas QM2. Tiene un canal de conexión de servidor de `QM2.SVRCONN`, un escucha en el puerto 1415y se ejecuta en el host `QM2.example.com`. Sin embargo, como las conexiones forman parte del grupo de gestores de colas QMGRP, el campo **queueManager** para ambas conexiones se establece en el nombre del grupo de gestores de colas.

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM1.example.com",
            "port": 1414
          }],
        "queueManager": "QMGRP"
      },
```

```
      "type": "clientConnection"
   }],
   "channel": [{
      "name": "QM2.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM2.example.com",
            "port": 1415
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
    }]
}
```

6. En la instalación que ejecuta el servidor mqweb que aloja el messaging REST API, utilice el mandato **setmqweb remote** para añadir el grupo de gestores de colas a la configuración del servidor mqweb.

   Como mínimo, debe especificar los parámetros siguientes:

   - **-qmgrName**, donde especifica el nombre de grupo para el grupo de gestores de colas.
   - **-ccdtURL**, donde especifica el URL de CCDT para los gestores de colas.
   - **-uniqueName**, donde especifica un nombre exclusivo para identificar el grupo de gestores de colas.
   - **-group**, para establecer la información del gestor de colas como si fuera para un grupo.

   Puede especificar otras opciones, como el nombre de usuario y la contraseña que se van a utilizar para la conexión, o detalles del almacén de confianza y del almacén de claves. Para obtener una lista completa de los parámetros que se pueden especificar con el mandato **setmqweb remote** , consulte setmqweb remote.

   Por ejemplo, para añadir el grupo de gestores de colas QMGRP, con el archivo CCDT de ejemplo, especifique el mandato siguiente:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

Resultados

El grupo de gestores de colas remotos se puede utilizar con messaging REST API utilizando el nombre exclusivo en el recurso URL. Se selecciona un gestor de colas del grupo para completar la solicitud, y la información sobre qué gestor de colas ha completado la solicitud se devuelve en la cabecera de respuesta `ibm-mq-resolved-qmgr`.

V 9.3.3 Determinación del principal de seguridad utilizado por messaging REST API

Cuando utilice messaging REST API, un usuario adecuado debe tener autorización para acceder a los gestores de colas, colas y temas a los que desea conectarse para la mensajería. El usuario que necesita autorización depende de cómo esté configurado el servidor mqweb y de si está utilizando gestores de colas remotos con messaging REST API.

De forma predeterminada, el principal de seguridad que se utiliza para autorizar el acceso al gestor de colas es el usuario que inicia el servidor mqweb que ejecuta el messaging REST API. El principal de seguridad que se utiliza para autorizar el acceso a las colas y temas es el usuario que ha iniciado la sesión en messaging REST API. Sin embargo, es posible que el servidor mqweb o la conexión del gestor de colas remoto estén configurados de forma que se utilice un principal de seguridad diferente.

Determinación del principal de seguridad que se utiliza para conectarse al gestor de colas

Para las conexiones de gestor de colas local, el principal de seguridad que se utiliza para conectarse al gestor de colas es el usuario que inicia el servidor mqweb que ejecuta messaging REST API. Para las conexiones de gestor de colas remoto, messaging REST API utiliza los siguientes principales de seguridad para autorizar el acceso al gestor de colas, por orden de prioridad. Es decir, si los usuarios se especifican

de varias maneras dentro de la configuración del gestor de colas remoto, el primero de la lista se utiliza para la autorización.

1. El principal de seguridad es un contexto de usuario adoptado de una salida de seguridad.
2. El principal de seguridad es un contexto de usuario adoptado en una regla CHLAUTH en el canal de conexión con el servidor que se utiliza para conectarse al gestor de colas remoto.
3. El principal de seguridad es el ID de usuario que se incluye en la configuración del gestor de colas remoto para messaging REST API. Este ID de usuario se incluye opcionalmente en la información de conexión del gestor de colas cuando se añade el gestor de colas con el mandato **setmqweb remote** .
4. El principal de seguridad es el usuario que inicia el servidor mqweb que ejecuta messaging REST API.

Para obtener más información sobre cómo configurar gestores de colas remotos para utilizarlos con messaging REST API, consulte "Configuración de un gestor de colas remoto para utilizarlo con messaging REST API" en la página 728.

Determinación del principal de seguridad que se utiliza para conectarse a colas y temas

Puede establecer una propiedad en la configuración del servidor mqweb para determinar qué principal de seguridad se utiliza para autorizar conexiones a colas y temas cuando utiliza messaging REST API. Esta propiedad es la propiedad **mqRestMessagingAdoptWebUserContext** . Puede ver en qué se establece esta propiedad utilizando el mandato **dspmqweb properties** .

- Si **mqRestMessagingAdoptWebUserContext** se establece en true, el messaging REST API utiliza el ID de usuario del usuario que ha iniciado la sesión en el messaging REST API para la autorización. Por lo tanto, el ID de usuario o los ID de usuario que existen en la configuración del servidor mqweb para su uso con messaging REST API son los principales de seguridad que deben estar autorizados para acceder a las colas y temas.
- Si **mqRestMessagingAdoptWebUserContext** se establece en false, el messaging REST API utiliza el ID de usuario del usuario que ha iniciado el servidor mqweb que aloja el messaging REST API para la autorización. Por lo tanto, un ID de usuario que sea el mismo que el ID de usuario que inicia el servidor mqweb que aloja el messaging REST API debe tener autorización para acceder a las colas y los temas.

  Si las colas y los temas están en un gestor de colas remoto, el principal de seguridad que se utiliza para la autorización puede estar determinado por los valores de la configuración del gestor de colas. Se pueden utilizar los siguientes principales de seguridad, en orden de prioridad:

  1. El principal de seguridad es un contexto de usuario adoptado de una salida de seguridad.
  2. El principal de seguridad es un contexto de usuario adoptado en una regla CHLAUTH en el canal de conexión con el servidor que se utiliza para conectarse al gestor de colas remoto. Por ejemplo, puede configurar una regla CHLAUTH en el canal de conexión de servidor para utilizar el parámetro MCAUSER. A continuación, todas las conexiones se correlacionan con un ID de usuario que está autorizado a utilizar el gestor de colas.
  3. El principal de seguridad es un contexto de usuario adoptado de AUTHINFO del gestor de colas. Si el objeto AUTHINFO al que hace referencia el atributo CONNAUTH del gestor de colas está configurado para utilizar **ADOPTCTX(yes)**, el principal de seguridad que se utiliza para autorizar las conexiones con el gestor de colas también se utiliza para autorizar las colas y los temas. Por ejemplo, este principal de seguridad puede ser el ID de usuario que se incluye en la información de conexión del gestor de colas remoto como parte del mandato **setmqweb remote** .

Información relacionada

CHLAUTH
CONNAUTH
Propiedades de dspmqweb

Desarrollo de aplicaciones MQI con IBM MQ

IBM MQ proporciona soporte para C, Visual Basic, COBOL, Assembler, RPG, pTALy PL/I. Estos lenguajes de procedimiento utilizan la interfaz de cola de mensajes (MQI) para acceder a los servicios de colas de mensajes.

Para ver información detallada sobre cómo escribir sus propias aplicaciones en un lenguaje concreto, consulte los subtemas.

Para obtener una visión general de la interfaz de llamada para los lenguajes de procedimiento, consulte Descripciones de llamadas. Este tema contiene una lista de las llamadas MQI, y cada llamada le muestra cómo codificar las llamadas en cada uno de estos lenguajes.

IBM MQ proporciona archivos de definición de datos para ayudarle a escribir sus aplicaciones. Para ver una descripción completa, consulte “Archivos de definición de datos de IBM MQ” en la página 736.

Para ayudarle a elegir el lenguaje procedural en los que codificar sus programas, tenga en cuenta la longitud máxima de los mensajes que procesarán sus programas. Si sus programas sólo procesarán mensajes de un tamaño conocido, puede codificarlos en cualquiera de los lenguajes admitidos. Si no sabe la longitud máxima de los mensajes que los programas tendrán que procesar, el lenguaje que elija dependerá de si está desarrollando una aplicación CICS, IMS o por lotes:

IMS y por lotes
: Utiliza programas en C, PL/I o ensamblador para utilizar las facilidades que ofrecen estos lenguajes para obtener y liberar cantidades de memoria arbitrarias. También podría utilizar COBOL; pero use subrutinas de lenguaje ensamblador, PL/I o C para obtener y liberar almacenamiento.

CICS
: Utilice cualquier lenguaje admitido por CICS. La interfaz EXEC CICS proporciona las llamadas para la gestión de memoria, si fuera necesario.

Conceptos relacionados

“Aplicaciones orientadas a objetos” en la página 16
IBM MQ proporciona soporte para JMS, Java, C++ y .NET. Estos lenguajes e infraestructuras utilizan el mismo modelo de objeto de IBM MQ que proporciona clases que incluyen las mismas funciones que las llamadas y estructuras de IBM MQ.

Visión general técnica

“Conceptos de desarrollo de aplicaciones” en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

Referencia relacionada

Guía de consulta para el desarrollo de aplicaciones

Archivos de definición de datos de IBM MQ

IBM MQ proporciona archivos de definición de datos para ayudarle a escribir sus aplicaciones.

Los archivos de definición de datos se conocen también como:

Idioma	Definiciones de datos
C	Archivos include o de cabecera
Visual Basic	Archivos de módulo (sólo versiones de 32 bits)
COBOL	Archivos de copia
Ensamblador	Macros
PL/I	Archivos include

Los archivos de definición de datos para ayudarle a escribir salidas de canal se describen en Archivos COPY, de cabecera, de inclusión y de módulo de IBM MQ.

Los archivos de definición datos para ayudarle a escribir salidas de servicios de canal instalables se describen en “Salidas de usuario, salidas de API y servicios instalables de IBM MQ” en la página 960.

Para los archivos de definición de datos soportados en C++, consulte Utilización de C++.

IBM i

Para obtener los archivos de definición de datos soportados en RPG, consulte la publicación IBM i Application Programming Reference (ILE/RPG).

Los nombres de los archivos de definición de datos tienen el prefijo CMQ y un sufijo que determina el lenguaje de programación:

Sufijo	Idioma
a	Lenguaje ensamblador
b	Visual Basic
c	C
l	COBOL (sin valores inicializados)
p	PL/I
v	COBOL (con los valores predeterminados establecidos)

Biblioteca de instalación

El nombre **thlqual** es el cualificador de alto nivel de la biblioteca de instalación en z/OS.

En este tema se presentan los archivos de definición de datos de IBM MQ, bajo estas cabeceras:

- “Archivos include (de inclusión) del lenguaje C” en la página 737
- “Archivos de módulo Visual Basic” en la página 738
- “Archivos de copia COBOL” en la página 738
- z/OS “Macros en lenguaje ensamblador de System/390” en la página 739
- z/OS “Archivos de inclusión PL/I” en la página 739

Archivos include (de inclusión) del lenguaje C

Los archivos de inclusión de IBM MQ C se listan en los archivos de cabecera C. Se instalan en los directorios o las bibliotecas siguientes:

Plataforma	Directorio o biblioteca de instalación
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows Windows	*MQ_INSTALLATION_PATH*\Herramientas \c\include
z/OS z/OS	**thlqual**.SCSQC370

donde *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.

Nota: Para AIX and Linux, los archivos de inclusión están enlazados simbólicamente en `/usr/include`.

Para obtener más información acerca de la estructura de los directorios, consulte Planificación del soporte del sistema de archivos.

Archivos de módulo Visual Basic

IBM MQ for Windows proporciona cuatro archivos de módulo de Visual Basic.

Se listan en Archivos de módulo Visual Basic y se instalan en

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

Archivos de copia COBOL

Para COBOL, IBM MQ proporciona archivos de copia separados que contienen las constantes con nombre, y dos archivos de copia para cada una de las estructuras.

Hay dos archivos para cada estructura porque cada una se proporciona con y sin valores iniciales:

- En la sección WORKING-STORAGE SECTION de un programa COBOL, utilice los archivos que inicializan los campos de estructura en los valores predeterminados. Estas estructuras se definen en los archivos de copia que tienen nombres que llevan como sufijo la letra V (valores).
- En la sección LINKAGE SECTION de un programa COBOL, utilice las estructuras sin valores iniciales. Estas estructuras se definen en archivos de copia que tienen nombres que llevan como sufijo la letra L (enlace).

IBM i Los archivos de copia que contienen datos y definiciones de interfaz para IBM i se proporcionan para los programas ILE COBOL utilizando llamadas prototipo a la MQI. Los archivos están en QMQM/QCBLLESRC con nombres de miembro que tienen un sufijo L (para estructuras sin valores iniciales) o V (para estructuras con valores iniciales).

Los archivos de copia COBOL de IBM MQ se listan en archivos COPY de COBOL. Se instalan en los directorios siguientes:

Plataforma	Directorio o biblioteca de instalación
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (para Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (para IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Incluya en el programa únicamente aquellos archivos que necesite. Hágalo con una o varias sentencias COPY después de una declaración de nivel 01. Esto significa que puede incluir varias versiones de las estructuras en un programa si es necesario. Tenga en cuenta que CMQV es un archivo grande.

A continuación, se muestra un ejemplo de código COBOL para incluir el archivo de copia CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Cada declaración de estructura empieza por el elemento de nivel 01; puede declarar varias instancias de la estructura codificando la declaración de nivel 01 seguida por una sentencia COPY para copiar en el

resto de la declaración de la estructura. Para hacer referencia a la instancia apropiada, utilice la palabra clave IN.

A continuación, se muestra un ejemplo de código COBOL para incluir dos instancias de CMQMDV:

```
* 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.
```

Alinee las estructuras en límites de 4 bytes. Si utiliza la sentencia COPY para incluir una estructura a continuación de un elemento que no es el elemento de nivel 01, asegúrese de que la estructura sea un múltiplo de 4-bytes desde el principio del elemento de nivel 01. Si no lo hace, podría reducirse el rendimiento de la aplicación.

Las estructuras se describen en Tipos de datos utilizados en la MQI. Las descripciones de los campos en las estructuras muestran los nombres de los campos sin un prefijo. En programas COBOL, anteponga como prefijo de los nombres de campo el nombre de la estructura seguido de un guión, tal como se muestra en las declaraciones de COBOL. Los campos en la estructura de archivos de copia llevan prefijos de este tipo.

Los nombres de campos en las declaraciones de los archivos de copia de la estructura están en mayúsculas. También puede utilizar una combinación de mayúsculas y minúsculas. Por ejemplo, el campo *StrucId* de la estructura MQGMO se muestra como MQGMO-STRUCID en la declaración COBOL y en el archivo de copia.

Las estructuras con sufijo V se declaran con valores iniciales para todos los campos, por lo que es necesario establecer sólo aquellos campos en los que el valor necesario sea diferente del valor inicial.

Macros en lenguaje ensamblador de System/390

z/OS

IBM MQ for z/OS proporciona dos macros de ensamblador que contienen las constantes con nombre, y una macro para generar cada estructura.

Se listan en los Archivos COPY de ensamblador de z/OS y se instalan en **thlqual**.SCSQMACS.

Estas macros se invocan utilizando un código cómo éste:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

Archivos de inclusión PL/I

z/OS

IBM MQ for z/OS proporciona archivos de inclusión que contienen todas las definiciones que necesita al escribir aplicaciones de IBM MQ en PL/I.

Los archivos se listan en Archivos de inclusión PL/I y se instalan en el directorio **thlqual**.SCSQPLIC:

Incluya estos archivos en el programa si va a enlazar el apéndice de IBM MQ con el programa (consulte “Preparación del programa para su ejecución” en la página 1049). Incluya sólo CMQP si tiene la intención de enlazar las llamadas de IBM MQ dinámicamente (consulte “Llamada dinámica al apéndice IBM MQ” en la página 1055). El enlace dinámico sólo se puede realizar para los programas de proceso por lotes y IMS.

Desarrollo de una aplicación procedimental para encolamientos

Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

Utilice los enlaces siguientes para obtener más información sobre el desarrollo de aplicaciones:

- "Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
- "Conexión y desconexión de un gestor de colas" en la página 754
- "Apertura y cierre de objetos" en la página 761
- "Colocación de mensajes en una cola" en la página 772
- "Obtención de mensajes de una cola" en la página 788
- "Escritura de aplicaciones de publicación/suscripción" en la página 829
- "Consulta y establecimiento de atributos de objeto" en la página 871
- "Confirmación y restitución de unidades de trabajo" en la página 874
- "Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886
- "Cómo trabajar con clústeres y MQI" en la página 907
- z/OS "Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
- z/OS "IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72

Conceptos relacionados

"Conceptos de desarrollo de aplicaciones" en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

"Desarrollo de aplicaciones para IBM MQ" en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

"Consideraciones acerca del diseño de las aplicaciones IBM MQ" en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

"Desarrollo de aplicaciones procedimentales cliente" en la página 936
Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

"Creación de una aplicación procedimental" en la página 1025
Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

"Tratamiento de errores en un programa procedimental" en la página 1064
Esta información explica los errores asociados a las llamadas MQI de una aplicación cuando se efectúa una llamada o cuando el mensaje se entrega en su destino final.

Tareas relacionadas

"Utilización de programas procedimentales de ejemplo de IBM MQ" en la página 1085
Estos programas de ejemplo se escriben en lenguajes de procedimiento y muestran los usos típicos de la interfaz de cola de mensajes (MQI). Programas IBM MQ en distintas plataformas.

Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)

Conozca los componentes de la interfaz de colas de mensajes (MQI).

La interfaz de cola de mensajes consta de lo siguiente:

- *Llamadas* mediante las cuales los programas pueden acceder al gestor de colas y a sus recursos.
- *Estructuras* que los programas utilizan para pasar datos al gestor de colas y obtener datos del mismo.
- *Tipos de datos elementales* para pasar datos al gestor de colas y obtener datos del mismo.

z/OS IBM MQ for z/OS también proporciona:

- Dos llamadas adicionales a través de las cuales los programas por lotes z/OS pueden confirmar y restituir cambios.
- *Archivos de definición de datos* (conocidos a veces como archivos de copia, macros, archivos de inclusión o archivos de cabecera), que definen los valores de las constantes suministradas con IBM MQ for z/OS.
- *Programas de apéndice (stub)* para editar enlaces de las aplicaciones.
- Una suite de programas de ejemplo que muestran cómo utilizar MQI en la plataforma z/OS. Para obtener más información sobre estos ejemplos, consulte "Utilización de los programas de ejemplo para z/OS" en la página 1189.

IBM i IBM MQ for IBM i también proporciona:

- *Archivos de definición de datos* (conocidos a veces como archivos de copia, macros, archivos de inclusión o archivos de cabecera), que definen los valores de las constantes suministradas con IBM MQ for IBM i.
- Tres programas de apéndice para editar enlaces de las aplicaciones ILE C, ILE COBOL e ILE RPG.
- Una suite de programas de ejemplo que muestran cómo utilizar MQI en la plataforma IBM i.

Los sistemas AIX, Linux, and Windows también suministran:

- Llamadas a través de las cuales los programas de sistemas IBM MQ for AIX, Linux, and Windows pueden confirmar y restituir cambios.
- *Archivos de inclusión* que definen los valores de las constantes suministradas en estas plataformas.
- *Archivos de biblioteca* para enlazar las aplicaciones.
- Una suite de programas de ejemplo que muestran cómo utilizar MQI en estas plataformas. Para obtener más información sobre estos ejemplos, consulte "Utilización de los programas de ejemplo en Multiplataformas" en la página 1086.
- Código de ejemplo fuente y ejecutable para enlazar con los gestores de transacciones externos.

Utilice los enlaces siguientes para obtener más información sobre MQI:

- "llamadas MQI" en la página 742
- "Llamadas de punto de sincronización" en la página 743
- "Conversión de datos, tipos de datos, definiciones de datos y estructuras" en la página 744
- "Archivos de biblioteca y programas de apéndice de IBM MQ" en la página 744
- "Parámetros comunes a todas las llamadas" en la página 749
- "Especificación de búfers" en la página 750
- z/OS "Consideraciones acerca de los programas z/OS por lotes" en la página 750
- "Manejo de señales AIX and Linux" en la página 751

Conceptos relacionados

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Apertura y cierre de objetos" en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Confirmación y restitución de unidades de trabajo” en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

“IMS y las aplicaciones puente IMS en IBM MQ for z/OS” en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

llamadas MQI

Utilice esta información para obtener información sobre las llamadas en la interfaz de colas de mensajes (Message Queue Interface, MQI).

Las llamadas de la MQI se pueden agrupar de la forma siguiente:

MQCONN, MQCONNX y MQDISC
Utilice estas llamadas para conectar un programa con un gestor de colas (con o sin opciones) y desconectarlo del mismo. Si escribe programas CICS para z/OS, no es necesario que utilice estas llamadas. Sin embargo, se recomienda utilizarlas si se desea que la aplicación se utilice en otras plataformas.

MQOPEN y MQCLOSE
Utilice estas llamadas para abrir y cerrar un objeto como, por ejemplo, una cola.

MQPUT y MQPUT1
Utilice estas llamadas para colocar un mensaje en una cola.

MQGET
Utilice esta llamada para examinar mensajes en una cola o eliminarlos de la misma.

MQSUB, MQSUBRQ
Utilice estas llamadas para registrar una suscripción a un tema y para solicitar publicaciones que coincidan con la suscripción.

MQINQ
Utilice esta llamada para interrogar los atributos de un objeto.

MQSET
Utilice esta llamada para establecer algunos de los atributos de una cola. No se pueden establecer los atributos de otros tipos de objeto.

MQBEGIN, MQCMIT y MQBACK
Utilice estas llamadas cuando IBM MQ sea el coordinador de una unidad de trabajo. MQBEGIN inicia la unidad de trabajo. MQCMIT y MQBACK finalizan la unidad de trabajo, ya sea confirmando o retrotrayendo las actualizaciones realizadas durante la unidad de trabajo. IBM i Se utiliza el controlador de confirmaciones de IBM i para coordinar unidades de trabajo globales en IBM MQ for IBM i. Se utilizan los comandos nativos de control de confirmación, confirmación y retrotracción.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH
Utilice estas llamadas para crear un manejador de mensajes, para convertir un descriptor de mensajes en un búfer o un búfer en un descriptor de mensaje, y para borrar un descriptor de mensaje.

MQSETMP, MQINQMP, MQDLTMP
Utilice estas llamadas para establecer una propiedad de mensaje en un descriptor de mensaje, consultar una propiedad de mensaje y borrar una propiedad de un descriptor de mensaje.

MQCB, MQCB_FUNCTION, MQCTL
Utilice estas llamadas para registrar y controlar una función de devolución de llamada.

MQSTAT
Utilice esta llamada para recuperar información de estado de las operaciones de colocación asíncronas anteriores.

Consulte Descripciones de llamadas para obtener una descripción de las llamadas MQI.

Llamadas de punto de sincronización

Utilice este tema para obtener información sobre las llamadas de punto de sincronización en distintas plataformas.

Las llamadas de punto de sincronización están disponibles de la forma siguiente:

llamadas IBM MQ for z/OS

IBM MQ for z/OS proporciona las llamadas MQCMIT y MQBACK.

Utilice estas llamadas en los programas por lotes de z/OS para indicar al gestor de colas que todas las operaciones MQGET y MQPUT desde el último punto de sincronización serán permanentes (confirmadas) o se restituirán. Para confirmar y restituir los cambios en otros entornos:

CICS
Utilice mandatos como, por ejemplo, EXEC CICS SYNCPOINT y EXEC CICS SYNCPOINT ROLLBACK.

IMS
Utilice los recursos de punto de sincronización de IMS como, por ejemplo, GU (get unique) en las llamadas IOPCB, CHKP (punto de comprobación) y ROLB (retrotracción).

RRS
Utilice MQCMIT y MQBACK o SRRCMIT y SRRBACK, según corresponda. (Consulte “Servicios de gestión de transacciones y gestor de recursos recuperables” en la página 879).

Nota: SRRCMIT y SRRBACK son mandatos RRS nativos, no son llamadas MQI.

llamadas IBM i

IBM MQ for IBM i proporciona los mandatos MQCMIT y MQBACK. También puede utilizar los mandatos COMMIT y ROLLBACK de IBM i, o cualquier otro mandato o llamada que inicie los recursos de control de compromiso de IBM i (por ejemplo, EXEC CICS SYNCPOINT).

Llamadas de IBM MQ en plataformas AIX, Linux, and Windows

IBM MQ for AIX, Linux, and Windows proporciona las llamadas MQCMIT y MQBACK.

Utilice las llamadas de punto de sincronización en los programas para indicar al gestor de colas que todas las operaciones MQGET y MQPUT desde el último punto de sincronización serán permanentes (confirmadas) o se restituirán. Para confirmar y restituir los cambios en el entorno CICS, utilice mandatos como, por ejemplo, EXEC CICS SYNCPOINT y EXEC CICS SYNCPOINT ROLLBACK.

Conversión de datos, tipos de datos, definiciones de datos y estructuras

Use esta información para informarse sobre conversiones de datos, tipos de datos elementales, definiciones de datos de IBM MQ y estructuras cuando se usa la interfaz de cola de mensajes (MQI).

Conversión de datos

La llamada MQXCNVC (convertir caracteres) convierte los datos de carácter de un mensaje de un juego de caracteres a otro. Salvo en IBM MQ for z/OS, esta llamada solo se usa en una salida de conversión de datos.

Consulte MQXCNVC: conversión de caracteres para ver la sintaxis utilizada en la llamada MQXCNVC y “Escribir salidas de conversión de datos” en la página 1009 para obtener instrucciones sobre cómo escribir e invocar salidas de conversión de datos.

Tipos de datos elementales

En el caso de los lenguajes de programación soportados, la MQI proporciona tipos de datos elementales o campos no estructurados.

Estos tipos de datos se describen completamente en Tipos de datos elementales.

Definiciones de datos IBM MQ

z/OS IBM MQ for z/OS proporciona definiciones de datos en forma de archivos de copia de COBOL, macros de lenguaje de ensamblaje, un único archivo de inclusión PL/I, un único archivo de inclusión de lenguaje C y archivos de inclusión del lenguaje C++.

IBM i IBM MQ for IBM i proporciona definiciones de datos en forma de archivos de copias de COBOL, archivos de copias RPG, archivos de inclusión de lenguaje C y archivos de inclusión de lenguaje C++.

Los archivos de definición de datos proporcionados con IBM MQ contienen:

- Definiciones de todas las constantes y códigos de retorno IBM MQ
- Definiciones de las estructuras y tipos de datos IBM MQ
- Definiciones de constante para inicializar las estructuras.
- Prototipos de función para cada una de las llamadas (solo para PL/I y C).

Para obtener una descripción completa de los archivos de definiciones de datos IBM MQ, consulte “Archivos de definición de datos de IBM MQ” en la página 736.

Estructuras

Las estructuras que se usan en las llamadas MQI listadas en “llamadas MQI” en la página 742 se proporcionan en los archivos de definición de datos de cada lenguaje de programación soportado.
IBM i z/OS IBM MQ for z/OS y IBM MQ for IBM i proporcionan archivos que contienen constantes que se usan al completar algunos de los campos de dichas estructuras. Para obtener más información, consulte Definiciones de datos IBM MQ.

Consulte Tipos de datos de estructura para obtener un resumen de las estructuras.

Archivos de biblioteca y programas de apéndice de IBM MQ

Los programas de apéndice y archivos de biblioteca proporcionados se listan aquí por cada plataforma.

Para obtener más información sobre cómo utilizar los programas de apéndice y los archivos de biblioteca al crear una aplicación ejecutable, consulte “Creación de una aplicación procedimental” en la página 1025. Para obtener más información sobre cómo enlazar con archivos de biblioteca C++, consulte Utilización de C++ *IBM MQ Uso de C++*.

AIX *Archivos de biblioteca de IBM MQ for AIX*

En IBM MQ for AIX, debe enlazar el programa a los archivos de biblioteca MQI proporcionados para el entorno en el cual está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

En una aplicación sin hebras, enlace con una de las bibliotecas siguientes:

Tabla 106. Archivos de biblioteca para aplicaciones AIX sin hilos

Archivo de biblioteca	**Entorno**
libmqm.a	Servidor en C
libmqic.a y libmqm.a	Cliente en C
libmqmzf.a	Salidas de servicio instalable en C
libmqmxa.a	Interfaz XA de servidor
libmqmxa64.a	Interfaz XA de servidor alternativa
libmqcxa.a	Interfaz XA de cliente
libmqcxa64.a	Interfaz XA de cliente alternativa
libmqmcbrt.o	Soporte de la biblioteca de tiempo de ejecución de IBM MQ para Micro Focus COBOL
libmqmcb.a	Servidor para COBOL
libmqicb.a	Cliente para COBOL
libimqc23ia.a	Cliente para C++ (XLC 16)
libimqs23ia.a	Servidor para C++ (XLC 16)
V 9.3.5 libimqc23ca.a	Cliente para C++ (XLC 17)
V 9.3.5 libimqs23ca.a	Servidor para C++ (XLC 17)

V 9.3.5 Las bibliotecas que contienen "ia" se han creado con el compilador XLC 16, mientras que las bibliotecas con "ca" en el nombre se han creado con el compilador XLC 17.

En una aplicación con hebras, enlace con una de las bibliotecas siguientes:

Tabla 107. Archivos de biblioteca para aplicaciones AIX con hebras.

Tabla de dos columnas que lista los archivos de biblioteca y el entorno de cada uno de ellos.

Archivo de biblioteca	**Entorno**
libmqm_r.a	Servidor en C
libmqic_r.a y libmqm_r.a	Cliente en C
libmqmzf_r.a	Salidas de servicio instalable en C
libmqmxa_r.a	Interfaz XA de servidor
libmqmxa64_r.a	Interfaz XA de servidor alternativa
libmqcxa_r.a	Interfaz XA de cliente
libmqcxa64_r.a	Interfaz XA de cliente alternativa
libimqc23ia_r.a	Cliente para C++ (XLC 16)
libimqs23ia_r.a	Servidor para C++ (XLC 16)

Tabla 107. Archivos de biblioteca para aplicaciones AIX con hebras. Tabla de dos columnas que lista los archivos de biblioteca y el entorno de cada uno de ellos. *(continuación)*	
Archivo de biblioteca	**Entorno**
V 9.3.5 libimqc23ca_r.a	Cliente para C++ (XLC 17)
V 9.3.5 libimqs23ca_r.a	Servidor para C++ (XLC 17)

V 9.3.5 Las bibliotecas que contienen "ia" se han creado con el compilador XLC 16, mientras que las bibliotecas con "ca" en el nombre se han creado con el compilador XLC 17.

Nota: No se puede enlazar a más de una biblioteca. Es decir, no se puede enlazar a una biblioteca con hebras ni a una biblioteca sin hebras al mismo tiempo.

IBM i *Archivos de biblioteca de IBM MQ for IBM i*

En IBM MQ for IBM i, enlace el programa a los archivos de biblioteca MQI proporcionados para el entorno en el cual está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

En aplicaciones sin hebras:

Tabla 108. Archivos de biblioteca para aplicaciones IBM i sin hilos	
Archivo de biblioteca	**Entorno**
LIBMQM	Programa de servicio de servidor y cliente
LIBMQIC	Programa de servicio cliente
IMQB23I4	Programa de servicio base en C++
IMQS23I4	Programa de servicio de servidor en C++
LIBMQMZF	Salidas instalables en C

En una aplicación con hebras:

Tabla 109. Archivos de biblioteca para aplicaciones IBM i con hebras	
Archivo de biblioteca	**Entorno**
LIBMQM_R	Programa de servicio de servidor y cliente
IMQB23I4_R	Programa de servicio base en C++
IMQS23I4_R	Programa de servicio de servidor en C++
LIBMQMZF_R	Salidas instalables en C
LIBMQIC_R	Programa de servicio cliente

En IBM MQ for IBM i, puede escribir las aplicaciones en C++. Para ver cómo enlazar las aplicaciones C++ y para obtener detalles completos de todos los aspectos del uso de C++, consulte Utilización de C++.

Linux *Archivos de biblioteca de IBM MQ para Linux*

En IBM MQ para Linux, debe enlazar el programa a los archivos de biblioteca MQI proporcionados para el entorno, en el cual está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

En una aplicación sin hebras, enlace con una de las bibliotecas siguientes:

Tabla 110. Archivos de biblioteca para aplicaciones Linux sin hilos	
Archivo de biblioteca	**Entorno**
libmqm.so	Servidor en C
libmqic.so y libmqm.so	Cliente en C
libmqmzf.so	Salidas de servicio instalable en C
libmqmxa.so	Interfaz XA de servidor
libmqmxa64.so	Interfaz XA de servidor alternativa
libmqcxa.so	Interfaz XA de cliente
libmqcxa64.so	Interfaz XA de cliente alternativa
libimqc23gl.so	Cliente en C++
libimqs23gl.so	Servidor en C++

En una aplicación con hebras, enlace con una de las bibliotecas siguientes:

Tabla 111. Archivos de biblioteca para aplicaciones Linux con hebras	
Archivo de biblioteca	**Entorno**
libmqm_r.so	Servidor en C
libmqic_r.so y libmqm_r.so	Cliente en C
libmqmzf_r.so	Salidas de servicio instalable en C
libmqmxa_r.so	Interfaz XA de servidor
libmqmxa64_r.so	Interfaz XA de servidor alternativa
libmqcxa_r.so	Interfaz XA de cliente
libmqcxa64_r.so	Interfaz XA de cliente alternativa
libimqc23gl_r.so	Cliente en C++
libimqs23gl_r.so	Servidor en C++

Nota: No se puede enlazar a más de una biblioteca. Es decir, no se puede enlazar a una biblioteca con hebras ni a una biblioteca sin hebras al mismo tiempo.

Windows *Archivos de biblioteca de IBM MQ for Windows*

En IBM MQ for Windows, debe enlazar el programa a los archivos de biblioteca de MQI proporcionados para el entorno en el que esté ejecutando la aplicación, además de los proporcionados por el sistema operativo:

Tabla 112. Archivos de biblioteca para aplicaciones Windows	
Archivo de biblioteca	**Entorno**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Servidor para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Cliente para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Interfaz XA de servidor para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Interfaz XA de cliente para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Cliente MTS para C (32 bits)

Tabla 112. Archivos de biblioteca para aplicaciones Windows (continuación)	
Archivo de biblioteca	**Entorno**
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	Soporte de TXSeries CICS servidor para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	Soporte de TXSeries CICS cliente para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Salidas de servicios instalables para C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Servidor para IBM COBOL (32-bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Servidor para Micro Focus COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Cliente para IBM COBOL (32-bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Cliente para Micro Focus COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Servidor para C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Cliente para C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Base para C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Cliente para C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Servidor para C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Cliente para C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Interfaz XA de servidor para C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	Interfaz XA de cliente para C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	MTS cliente para C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Servidor para IBM COBOL (64-bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Servidor para Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Cliente para IBM COBOL (64-bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Cliente para Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Servidor para C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Cliente para C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Base para C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Cliente MTS para C++ (64 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Utilice `amqmdnet.dll` para compilar programas .NET. Consulte “compilar programas IBM MQ .NET” en la página 629 en la sección “Desarrollo de aplicaciones .NET” en la página 571 para obtener más información.

Estos archivos se suministran por motivos de compatibilidad con releases anteriores:

```
mqic32.lib
mqic32xa.lib
```

z/OS *Programas de apéndice de IBM MQ for z/OS*

Antes de poder ejecutar un programa escrito con IBM MQ for z/OS, debe editarlo mediante un enlace al programa de apéndice proporcionado con IBM MQ for z/OS para el entorno en el cual está ejecutando la aplicación.

El programa de apéndice proporciona la primera etapa del proceso de las llamadas en las solicitudes que puede procesar IBM MQ for z/OS.

IBM MQ for z/OS proporciona los programas de apéndice siguientes:

CSQBSTUB
Programa de apéndica para programas por lotes de z/OS

CSQBRRSI
Programa de apéndica para programas por lotes de z/OS utilizando RRS mediante la MQI

CSQBRSTB
Programa de apéndice para programas por lotes de z/OS utilizando RRS directamente

CSQCSTUB
Programa de apéndice para programas CICS

CSQQSTUB
Programa de apéndice para programas IMS

CSQXSTUB
Programa de apéndice para las salidas no CICS de agrupación en colas distribuidas

CSQASTUB
Programa de apéndice para salidas de conversión de datos

Atención: Si se usa un programa de apéndice distinto del listado para un determinado entorno, podría arrojar resultados imprevistos.

Nota: Si se usa el programa de apéndice CSQBRSTB, enlácelo-edítelo con ATRSCSS de SYS1.CSSLIB. (SYS1.CSSLIB también se conoce como la *Biblioteca de servicios invocables*). Para obtener más información sobre RRS, consulte "Servicios de gestión de transacciones y gestor de recursos recuperables" en la página 879.

De forma alternativa, se puede invocar dinámicamente el apéndice desde el programa. Esta técnica se describe en "Llamada dinámica al apéndice IBM MQ" en la página 1055.

En IMS, también podría necesitar utilizar un módulo de interfaz de lenguaje especial proporcionado por IBM MQ.

No ejecute las aplicaciones que están enlazadas-editadas con CSQBSTUB y CSQQSTUB en la misma región MPP de IMS. Esto puede provocar problemas como, por ejemplo, mensajes DFS3607I o CSQQ005E. La primera llamada MQCONN en un espacio de direcciones determina qué interfaz se utiliza, por lo tanto, las transacciones CSQQSTUB y CSQBSTUB se deben ejecutar en distintas regiones de mensajes de IMS.

Parámetros comunes a todas las llamadas

Existen dos tipos de parámetro comunes a todas las llamadas: descriptores y códigos de retorno.

Uso de un descriptor

Todas las llamadas MQI utilizan uno o más *descriptores*. Estos identifican un gestor de colas, una cola u otro objeto, mensaje o suscripción, según corresponda en la llamada.

Para que un programa se comunique con un gestor de colas, aquel habrá de tener un identificador exclusivo que le permita referenciar dicho gestor de colas. Este identificador se llama *descriptor de conexión*, referido a veces como *Hconn*. Para los programas CICS, el descriptor de conexión siempre es cero. En todas las demás plataformas o estilos de programas, el descriptor de conexión siempre lo devuelve una llamada MQCONN o MQCONNX cuando el programa se conecta con el gestor de colas. Los programas pasan el descriptor de conexión como un parámetro de entrada de otras llamadas.

Para que un programa funcione con un objeto IBM MQ, el programa debe tener un identificador exclusivo con el que referenciar dicho objeto. Este identificador se llama *descriptor de objeto*, referido a veces como *Hobj*. Este lo devuelve una llamada MQOPEN cuando el programa abre el objeto para trabajar con el. Los programas pasan el descriptor de objeto como parámetro de entrada cuando utilizan llamadas MQPUT, MQGET, MQINQ, MQSET o MQCLOSE posteriores.

De forma similar, la llamada MQSUB devuelve un *descriptor de suscripción* o *Hsub* que se utiliza para identificar la suscripción en las llamadas MQGET, MQCB o MQSUBRQ posteriores y determinadas llamadas que procesan propiedades de mensajes usan un *descriptor de mensaje* o *Hmsg*.

El código de retorno

Cada llamada devuelve un código de terminación y un código de razón en forma de parámetros de salida. Éstos se conocen de forma general como *códigos de retorno*.

Para mostrar si ha sido satisfactoria, cada llamada devuelve un *código de terminación* cuando termina la llamada. El código de terminación suele ser MQCC_OK, que indica que todo ha ido bien, o MQCC_FAILED, que indica un error. Algunas llamadas pueden devolver un estado intermedio, MQCC_WARNING, lo que indica un éxito parcial.

Cada llamada también devuelve un *código de razón* que muestra la razón del fallo éxito parcial de la llamada. Hay muchos códigos de razón que abarcan situaciones tales como una cola que está llena, operaciones de obtención no permitidas por una cola y una determinada cola que no está definida en el gestor de colas. Los programas pueden utilizar el código de razón para decidir cómo proceder. Por ejemplo, pueden solicitar al usuario que cambie sus datos de entrada yvuelvan a invocar la llamada, o bien puede devolverle un mensaje de error.

Cuando el código de terminación es MQCC_OK, el código de razón es siempre MQRC_NONE.

Los códigos de terminación y de razón de cada llamada se listan en la descripción de dicha llamada. Consulte Descripciones de llamadas y seleccione la corespondiente llamada en la lista.

Para obtener información detallada, junto con ideas para una acción correctiva, consulte:

- z/OS IBM MQ for z/OS mensajes, finalización, y códigos de razón para IBM MQ for z/OS
- Mensajes y códigos de razón para todas las demás plataformas IBM MQ

Especificación de búfers

El gestor de colas solo referencia un búfer cuando es necesario. Si no se necesita un búfer en una llamada o este tiene una longitud cero,se puede utilizar un puntero nulo a un búfer.

Utilice siempre datalength (longitud de datos) cuando especifique el tamaño del búfer que necesita.

Cuando se utiliza un búfer para alojar la salida de una llamada (por ejemplo, para alojar los datos de mensaje de una llamada MQGET o los valores de atributos consultados por la llamada MQINQ), el gestor de colas intenta devolver un código de razón si el búfer especificado no es válido o está en almacenamiento de solo lectura. No obstante, puede que no siempre sea capaz de devolver un código de razón.

z/OS *Consideraciones acerca de los programas z/OS por lotes*

Los programas z/OS por lotes que invocan MQI puede estar en estado de supervisor o de problema.

No obstante, debe cumplir las condiciones siguientes:

- Debe estar en modo de tarea y no en modo de bloque de solicitud de servicio (SRB).
- Debe estar en modo de control de espacio de direcciones (ASC) primario y no en modo de ASC de registro de acceso.
- No deben estar en modo de memoria cruzada. El número del espacio de direcciones (ASN) primario debe ser igual al ASN secundario y el ASN de inicio.
- No se deben utilizar como programas de salida MPF.
- No se debe mantener ningún bloqueo de z/OS.

- No pueden haber rutinas de recuperación de función (FRR) en la pila FRR.
- Para la llamada MQCONN o MQCONNX puede estar en vigor cualquier clave de palabra de estado de programa (PSW), siempre que la clave sea compatible con el uso del almacenamiento incluido en la clave TCB, pero las llamadas posteriores que devuelve MQCONN o MQCONNX:
  - Deben tener la misma clave PSW que se ha utilizado en llamada MQCONN o MQCONNX
  - Deben tener parámetros accesibles, para escritura si se requiere, bajo la misma clave PSW
  - Se deben emitir bajo la misma tarea (TCB), pero no así en ninguna subtarea de la tarea.
- Puede estar en modo de direccionamiento de 24 bits o de 31 bits. No obstante, si está en vigor el modo de direccionamiento de 24 bits, las direcciones de los parámetros se deben interpretar como direcciones de 31 bits válidas.

Si no se cumple cualquiera de estas condiciones, es posible que se produzca una comprobación del programa. En algunos casos, la llamada falla y se devuelve un código de razón.

Linux AIX *Consideraciones sobre AIX and Linux*

Consideraciones que debe tener en cuenta al desarrollar aplicaciones AIX and Linux .

Linux AIX *La llamada de sistema fork en sistemas AIX and Linux*

Tenga en cuenta estas consideraciones cuando utilice una llamada al sistema fork en aplicaciones IBM MQ.

Si una aplicación necesita usar `fork`, su proceso padre tendrá que invocar `fork` antes de que se efectúe cualquier llamada IBM MQ, por ejemplo, `MQCONN` o se cree un objeto IBM MQ usando **`ImqQueueManager`**.

Si la aplicación necesita crear un proceso hijo tras efectuar una llamada IBM MQ, el código de la misma tendrá que usar un `fork()` con `exec()` para asegurarse de que el hijo sea una instancia nueva y no una copia exacta del padre.

Si la aplicación no utiliza `exec()`, la llamada de API IBM MQ realizada en el proceso hijo devuelve MQRC_ENVIRONMENT_ERROR.

Linux AIX *Manejo de señales AIX and Linux*

En general, los sistemas AIX and Linux se han movido de un entorno sin hebras (proceso) a un entorno multihebra. En muchos casos, las señales y su manejo, aunque están soportados, no encajan bien en un entorno multihebra y existen varias restricciones.

En general, los sistemas AIX and Linux se han movido de un entorno sin hebras (proceso) a un entorno multihebra. En el entorno sin hebras, algunas funciones solo se pueden implementar utilizando señales, aunque la mayoría de las aplicaciones no necesitan tener conocimiento de las señales y su manejo. En un entorno multihebra, las primitivas basadas en hebras soportan algunas de las funciones que en los entornos sin hebras se solían implementar mediante señales.

En muchos casos, las señales y su manejo, aunque están soportados, no encajan bien en un entorno multihebra y existen varias restricciones. Esto puede ser problemático cuando se está integrando código de aplicación con diferentes bibliotecas de middleware (que ejecutan como parte de la aplicación) en un entorno multihebra donde cada una está intentando manejar las señales. El enfoque tradicional de guardar y restaurar los manejadores de señales (definidos a nivel de proceso), que funcionaba cuando solo había una única hebra de ejecución dentro de un proceso, no funciona en un entorno multihebra. Esto se debe a que muchas hebras de ejecución podrían estar intentando guardar y restaurar un recurso a nivel de proceso, con resultados impredecibles.

Linux AIX *Aplicaciones sin hebras*

Cada función MQI configura su propio manejador de señales para las señales. Los manejadores de los usuarios para estas se sustituyen mientras dure la llamada de función MQI. Los manejadores escritos por los usuarios pueden capturar las demás señales de la forma habitual.

Cada función de MQI establece su propio manejador de señales para estas señales:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

Los manejadores de los usuarios para estas se sustituyen mientras dure la llamada de función MQI. Los manejadores escritos por los usuarios pueden capturar las demás señales de la forma habitual. Si no se instala un manejador, las acciones predeterminadas (por ejemplo, ignorar, volcar el núcleo o salir) se dejan en su lugar.

Una vez que IBM MQ maneja una señal síncrona (SIGSEGV, SIGBUS, SIGFPE, SIGILL), intenta pasar la señal a cualquier manejador de señales registrado antes de realizar la llamada a la función MQI.

Linux AIX *Aplicaciones con hebras*

Se considera que una hebra está conectada a IBM MQ desde MQCONN (o MQCONNX) hasta MQDISC.

Señales síncronas

Una señal síncrona surge en una hebra concreta.

Los sistemas AIX and Linux permiten una configuración segura de un manejador de señal para dichas señales para todo el proceso. Si embargo, IBM MQ configura su propio manejador para las señales siguientes, en el proceso de aplicación, mientras que haya alguna hebra conectada a IBM MQ:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Si está escribiendo aplicaciones multihebra, solo hay un manejador de señales a nivel de proceso por cada señal. Cuando IBM MQ configura sus propios manejadores de señales síncronas, guardar los manejadores registrados previamente para cada señal. Después de que IBM MQ maneje una de las señales listadas, IBM MQ intenta llamar al manejador de señales que estaba en vigor en el momento de la primera conexión IBM MQ dentro del proceso. Los manejadores registrados previamente se restauran cuando todas las hebras de aplicación se hayan desconectado de IBM MQ.

Puesto que los manejadores de señales se guardan y restauran mediante IBM MQ, las hebras de aplicación no deben establecer manejadores de señales para estas señales, mientras exista la posibilidad de que otra hebra del mismo proceso también esté conectada a IBM MQ.

Nota: Cuando una aplicación o una biblioteca de middleware (que ejecute como parte de una aplicación) establece un manejador de señales mientras una hebra está conectada con IBM MQ, el manejador de señales de la aplicación tiene que invocar el correspondiente manejador de IBM MQ durante el procesamiento de dicha señal.

Al establecer y restaurar manejadores de señales, el principio general es que el último manejador de señales que se guarda tiene que ser el primero que se restaura:

- Cuando una aplicación establece un manejador de señales después de conectarse a IBM MQ, el manejador de señales anterior debe restaurarse antes de que la aplicación se desconecte de IBM MQ.
- Cuando una aplicación establece un manejador de señales antes de conectarse a IBM MQ, la aplicación debe desconectarse de IBM MQ antes de restaurar su manejador de señales.

Nota: Si no se respeta el principio general de que el último manejador de señales que se guarde tiene que ser el primero que se restaure, se puede dar lugar a un tratamiento de señales inesperado en la aplicación y, potencialmente, a que la esta pierda señales.

Señales asíncronas

IBM MQ no utilice ninguna señal asíncrona en las aplicaciones con hebras, a menos que sean aplicaciones cliente.

Consideraciones adicionales sobre las aplicaciones cliente con hebras

IBM MQ maneja las señales siguientes durante la E/S de un servidor. La pila de comunicaciones define estas señales. La aplicación no debe establecer un manejador de señales para estas señales mientras una hebra esté conectada con un gestor de colas:

SIGPIPE (en TCP/IP)

Consideraciones adicionales cuando se utiliza el manejo de señales AIX and Linux en MQI

Cuando se utiliza MQI para el manejo de señales en AIX and Linux, existen consideraciones adicionales para aplicaciones de vía de acceso rápida, llamadas de función MQI dentro de manejadores de señales, señales durante llamadas MQI, salidas de usuario y servicios instalables y manejadores de salida VMS.

Aplicaciones Fastpath (de confianza)

Las aplicaciones Fastpath ejecutan en el mismo proceso que IBM MQ y, por lo tanto, se ejecutan en el entorno multihebra.

En este entorno, IBM MQ maneja las señales síncronas SIGSEGV, SIGBUS, SIGFPE y SIGILL. Todas las demás señales no se deben entregar a la aplicación Fastpath mientras esté conectado a IBM MQ. En vez de ello, la aplicación tiene que bloquearlas o tratarlas. Si una aplicación Fastpath intercepta un suceso de este tipo, habrá que parar y reiniciar el gestor de colas o podría quedar en un estado indefinido. Para obtener una lista completa de las restricciones para las aplicaciones Fastpath bajo MQCONNX, consulte “Conexión a un gestor de colas mediante la llamada MQCONNX” en la página 756.

Llamadas de función MQI dentro de un manejador de señal

Mientras se encuentre en un manejador de señal, no invoque una función MQI.

Si intenta invocar una función MQI desde un manejador de señal mientras otra función MQI está activa, se devuelve MQRC_CALL_IN_PROGRESS. Si se intenta invocar una función MQI desde un manejador de señal mientras no hay ninguna otra función MQI activa, es probable que falle en algún momento durante la operación debido a las restricciones del sistema operativo por las que solo se pueden emitir llamadas selectivas desde un manejador.

En los métodos de destructor C++, que se pueden llamar automáticamente durante la salida de un programa, podría ser imposible impedir la llamada de funciones MQI. Ignore los errores relativos a MQRC_CALL_IN_PROGRESS. Si un manejador de señales llama a exit(), IBM MQ restituye todos los mensajes sin confirmar en el punto de sincronización de forma habitual y cierra las colas abiertas.

Señales durante llamadas MQI

Las funciones MQI no devuelven el código EINTR ni cualquier código equivalente a los programas de aplicación.

Si se produce una señal durante una llamada MQI y el manejador invoca *return*, la llamada seguirá ejecutándose como si la señal no se hubiera producido. En particular, MQGET no se puede interrumpir con una señal para devolver el control de forma inmediata a la aplicación. Si desea salir de un MQGET, establezca la cola a GET_DISABLED; de forma alternativa, utilice un bucle alrededor de una llamada a MQGET con un tiempo de caducidad finito (MQGMO_WAIT con gmo.WaitInterval) y utilice el manejador de señal (en un entorno sin hebras) o una función equivalente en un entorno con hebras para establecer un distintivo que interrumpa el bucle.

AIX En el entorno AIX, IBM MQ requiere que se reinicien las llamadas del sistema interrumpidas por las señales. Al establecer su propio manejador de señales con sigaction(2), establezca el distintivo SA_RESTART en el campo sa_flags de la nueva estructura de acción, de lo contrario IBM MQ podría no ser capaz de completar ninguna llamada interrumpida por una señal.

Salidas de usuario y servicios instalables

Las salidas de usuario y los servicios instalables que se ejecutan como parte de un proceso IBM MQ en un entorno de varias hebras tienen las mismas restricciones que para las aplicaciones Fastpath. Considérelas como permanentemente conectadas con IBM MQ de forma que no usen señales o llamadas de sistema operativo que no sean de hebra segura (thread-safe).

Conexión y desconexión de un gestor de colas

Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

La manera en que se realiza esta conexión depende de la plataforma y del entorno en los que opera el programa:

Multi **IBM MQ for Multiplatforms**

Los programas que se ejecutan en estos entornos pueden utilizar las llamadas MQI MQCONN y MQDISC para conectarse y desconectarse respectivamente de un gestor de colas. Como alternativa, los programas pueden utilizar la llamada MQCONNX.

z/OS **Lote IBM MQ for z/OS**

Los programas que se ejecutan en este entorno pueden utilizar las llamadas MQI MQCONN y MQDISC para conectarse y desconectarse respectivamente de un gestor de colas. Como alternativa, los programas pueden utilizar la llamada MQCONNX.

Los programas por lotes de z/OS pueden conectarse, de forma consecutiva o simultánea, a varios gestores de colas en el mismo TCB.

z/OS **IMS**

La región de control IMS se conecta a uno o más gestores de colas cuando se inicia. Esta conexión se controla mediante mandatos IMS. Para obtener información sobre cómo controlar el adaptador IMS en z/OS, consulte Administración IBM MQ for z/OS. No obstante, los creadores de programas IMS de colas de mensajes deben utilizar la llamada MQI MQCONN para especificar el gestor de colas al que se desean conectar. Pueden utilizar la llamada MQDISC para desconectarse de dicho gestor de colas.

Tras una llamada IMS que establece un punto de sincronización y antes de procesar un mensaje para otro uso, el adaptador IMS se asegura de que la aplicación cierre los manejadores y se desconecte del gestor de colas. Consulte “Puntos de sincronismo en aplicaciones IMS” en la página 878

Los programas IMS pueden conectarse, de forma consecutiva o simultánea, a varios gestores de colas en el mismo TCB.

z/OS **CICS Transaction Server para z/OS**

Los programas CICS no necesitan realizar ningún trabajo para conectarse a un gestor de colas, ya que el propio sistema CICS está conectado. Esta conexión se establece normalmente de forma automática en la inicialización, pero también puede utilizar la transacción CKQC que se proporciona con IBM MQ for z/OS. Para obtener más información sobre CKQC, consulte Administración IBM MQ for z/OS.

Las tareas CICS se pueden conectar únicamente al gestor de colas al que esté conectada la región CICS.

Los programas CICS también pueden utilizar las llamadas MQI de conexión y desconexión (MQCONN y MQDISC). Es posible que desee hacer esto para poder trasladar estas aplicaciones a entornos que no sean CICS con una recodificación mínima. No obstante, estas llamadas *siempre* se completan correctamente en un entorno CICS. Esto implica que es posible que el código de retorno no refleje el verdadero estado de la conexión con el gestor de colas.

TXSeries para Windows y Open Systems

Estos programas no necesitan realizar ningún trabajo para conectarse a un gestor de colas, ya que el propio sistema CICS está conectado. Por lo tanto, solo se admite una conexión al mismo tiempo. Las aplicaciones CICS deben emitir una llamada MQCONN para obtener un descriptor de conexión y una llamada MQDISC antes de salir.

Utilice los enlaces siguientes para obtener más información sobre la conexión y desconexión de un gestor de colas:

- “Conexión con un gestor de colas usando la llamada MQCONN” en la página 755
- “Conexión a un gestor de colas mediante la llamada MQCONNX” en la página 756
- “Desconectar programas desde un gestor de colas utilizando MQDISC” en la página 760

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Confirmación y restitución de unidades de trabajo” en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

“IMS y las aplicaciones puente IMS en IBM MQ for z/OS” en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Conexión con un gestor de colas usando la llamada MQCONN

Utilice esta información para obtener información sobre cómo conectarse con un gestor de colas usando la llamada MQCONN.

En general, se puede conectar con un gestor de colas concreto o bien con el gestor de colas predeterminado:

- Para IBM MQ for z/OS, en el entorno de proceso por lotes, el gestor de colas predeterminado es especifica en el módulo CSQBDEFV.
- Para IBM MQ for Multiplatforms, el gestor de colas predeterminado se especifica en el archivo mqs.ini .

De forma alternativa, en los entornos de proceso por lotes MVS, TSO y RRS de z/OS, puede conectarse a cualquier gestor de colas dentro de un grupo de compartición de colas. La solicitud MQCONN o MQCONNX selecciona cualquiera de los miembros activos del grupo.

Cuando se conecta con un gestor de colas, este tiene que ser local a la tarea. Debe pertenecer al mismo sistema que la aplicación IBM MQ.

En el entorno IMS, el gestor de colas tiene que estar conectado a la región de control de IMS y a la región dependiente que utiliza el programa. El gestor de colas predeterminado se especifica en el módulo CSQQDEFV cuando se instala IBM MQ for z/OS.

En los entornos TXSeries CICS, y en TXSeries para Windows y AIX, el gestor de colas tiene que definirse como un recurso XA a CICS.

Para conectarse con el gestor de colas predeterminado, llame MQCONN especificando un nombre que conste enteramente de espacios en blanco o que empiece por un carácter nulo (X'00 ').

Un aplicación tiene que estar autorizada para conectarse satisfactoriamente con un gestor de colas. Puede obtener información adicional consultando Protección.

La salida de MQCONN es:

- Un descriptor de conexión (**Hconn**).
- Un código de terminación
- Un código de razón

Utilice el descriptor de conexión en las llamadas MQI posteriores.

Si el código de razón indica que la aplicación ya está conectada con ese gestor de colas, el descriptor de conexión devuelto será el mismo que el devuelto cuando la aplicación se conectó por primera vez. La aplicación no puede emitir la llamada MQDISC en esta situación porque la aplicación invocadora espera seguir conectada.

El ámbito del descriptor de conexión es el mismo que el del descriptor de objeto (consulte “Abrir objetos con la llamada MQOPEN” en la página 763).

Las descripciones de los parámetros se proporcionan en la descripción de la llamada MQCONN en MQCONN.

La llamada MQCONN falla si el gestor de colas se encuentra en estado de desactivación temporal cuando se emite la llamada, o si el gestor de colas se está cerrando.

Ámbito de MQCONN o MQCONNX

El ámbito de una llamada MQCONN o MQCONNX suele ser la hebra que la ha emitido. Es decir, el descriptor de conexión que devuelve la llamada solo es válido dentro de la hebra que ha emitido dicha llamada. Solo se puede realizar una única llamada en cualquier momento utilizando el descriptor. Si se utiliza en una hebra distinta, se rechazará como no válida. Si tiene varias hebras en la aplicación y cada una de ellas desea utilizar llamadas IBM MQ, cada una debe emitir MQCONN o MQCONNX.

No es necesario que cada llamada se realice en el mismo gestor de colas cuando un proceso realiza varias llamadas MQCONN. Sin embargo, solo se puede realizar una conexión IBM MQ desde una hebra a la vez. De forma alternativa, considere “Conexiones (independientes de hebra) compartidas con MQCONNX” en la página 758 para permitir que se utilicen varias conexiones de IBM MQ desde una sola hebra y una conexión de IBM MQ desde cualquier hebra.[7]

Si la aplicación ejecuta como cliente, se puede conectar con más de un gestor de colas dentro de una hebra.

Conexión a un gestor de colas mediante la llamada MQCONNX

La llamada MQCONNX es parecida a la llamada MQCONN, pero incluye opciones para poder controlar la manera en que funciona la llamada.

[7] Al utilizar aplicaciones multihebra con sistemas IBM MQ for AIX or Linux , debe asegurarse de que las aplicaciones tienen un tamaño de pila suficiente para las hebras. Considere utilizar un tamaño de pila de 256 KB, o superior, cuando las aplicaciones con varias hebras estén realizando llamadas MQI, ya sea por sí mismas o con otros manejadores de señal (por ejemplo, CICS).

Como entrada en MQCONNX, puede proporcionar un nombre de gestor de colas z/OS, o un nombre de grupo de compartición de colas en z/OS sistemas de colas compartidas. Las opciones para controlar cómo se realiza la conexión con el gestor de colas se proporcionan en una estructura denominada MQCNO.

La salida de MQCONNX es:

- Un manejador de conexiones (Hconn)
- Un código de terminación
- Un código de razón

Puede utilizar el manejador de conexión en las llamadas MQI posteriores.

Las opciones de conexión, establecidas en el campo *Options* de la estructura MQCNO, permiten controlar varios atributos de la conexión. Cabe destacar los siguientes grupos de opciones:

- Las opciones de enlace permiten crear *aplicaciones de confianza* . Las aplicaciones de confianza implican que la aplicación IBM MQ y el agente del gestor de colas local se convierten en el mismo proceso. Dado que el proceso del agente ya no necesita utilizar una interfaz para acceder al gestor de colas, estas aplicaciones se convierten en una extensión del gestor de colas. Este comportamiento se solicita especificando la opción MQCNO_FASTPATH_BINDING . Para obtener más información sobre las restricciones que se aplican a las aplicaciones de confianza, consulte “Restricciones para aplicaciones de confianza” en la página 757.
- Las opciones de uso compartido de descriptor de contexto permiten crear conexiones compartidas. Las conexiones compartidas pueden compartir manejadores entre distintas hebras dentro del mismo proceso. Para obtener más información sobre las conexiones compartidas, consulte “Conexiones (independientes de hebra) compartidas con MQCONNX” en la página 758.

MQCNO también permite a la aplicación controlar cómo se autentica la conexión con el gestor de colas. Las credenciales de autenticación se pueden especificar en una estructura MQCSP a la que se hace referencia desde la estructura MQCNO.

Para obtener una descripción completa de los parámetros de la llamada MQCONNX y de los atributos de conexión que se pueden controlar, consulte Gestor de colas MQCONNX-Connect (ampliado).

Restricciones para aplicaciones de confianza
Restricciones que se aplican a las aplicaciones de confianza. Algunas restricciones se aplican a todas las plataformas y otras son específicas de la plataforma.

T

- Debe desconectar explícitamente las aplicaciones de confianza del gestor de colas.
- Debe detener las aplicaciones de confianza antes de finalizar el gestor de colas con el mandato **endmqm** .
- No debe utilizar señales asíncronas ni las interrupciones de temporizador (por ejemplo, `sigkill`) con MQCNO_FASTPATH_BINDING.
- En todas las plataformas, una hebra que esté dentro de una aplicación de confianza no puede conectarse a un gestor de colas mientras otra hebra del mismo proceso esté conectada a un gestor de colas diferente.
- Linux AIX En sistemas AIX and Linux, debe utilizar mqm como userID y groupID efectivos para todas las llamadas MQI. Puede cambiar estos ID antes de realizar una llamada que no sea MQI que requiera autenticación (por ejemplo, para abrir un archivo), pero debe volver a cambiarlos a mqm antes de realizar la siguiente llamada MQI.
- IBM i En IBM i:

  1. Las aplicaciones fiables deben ejecutarse bajo el perfil de usuario QMQM. No es suficiente que el perfil de usuario sea miembro del grupo QMQM, ni que el programa adopte la autorización QMQM. Puede que no sea posible utilizar el perfil de usuario QMQM para iniciar la sesión en los trabajos interactivos, ni que se especifique en la descripción de trabajo para los trabajos que

ejecuten aplicaciones fiables. En este caso, un método consiste en utilizar las funciones de API de intercambio de perfiles de IBM i , QSYGETPH, QWTSETP y QSYRLSPH para cambiar temporalmente el usuario actual del trabajo a QMQM mientras se ejecutan los programas IBM MQ . Los detalles de estas funciones, junto con un ejemplo de su uso, se proporcionan en la sección API de seguridad de la documentación de IBM i*Interfaces de programación de aplicaciones* .

2. No cancele las aplicaciones fiables mediante System-Request Option 2, ni finalizando los trabajos en ejecución que utilicen ENDJOB.

- ALW En los sistemas AIX, Linux, and Windows, no se admiten las aplicaciones de 32 bits de confianza. Si trata de ejecutar una aplicación de confianza de 32 bits, ésta se degradará a una conexión de vínculo estándar.

Conexiones (independientes de hebra) compartidas con MQCONNX
Utilice esta información para obtener información sobre las conexiones compartidas con MQCONNX, y algunas notas de uso que deben tenerse en cuenta.

Nota: No está soportado en IBM MQ for z/OS.

En plataformas IBM MQ distintas a IBM MQ for z/OS, una conexión realizada con MQCONN solo está disponible en la hebra que ha realizado la conexión. Las opciones de la llamada MQCONNX permiten crear una conexión que se puede compartir entre todas las hebras de un proceso. Si la aplicación ejecuta en un entorno transaccional que requiera emitir llamadas MQI en la misma hebra, habrá que utilizar la opción predeterminada siguiente:

MQCNO_HANDLE_SHARE_NONE
Crea una conexión no compartida.

En la mayoría de los demás entornos, se puede utilizar una de las siguientes opciones de conexión compartida independiente de hebra:

MQCNO_HANDLE_SHARE_BLOCK
Crea una conexión compartida. En una conexión MQCNO_HANDLE_SHARE_BLOCK, si una llamada MQI está usando en ese momento la conexión en otra hebra, la llamada MQI esperará a que la llamada MQI actual se haya completado.

MQCNO_HANDLE_SHARE_NO_BLOCK
Crea una conexión compartida. En una conexión MQCNO_HANDLE_SHARE_NO_BLOCK, si una llamada MQI utiliza actualmente la conexión en otra hebra, la llamada MQI falla inmediatamente con la razón MQRC_CALL_IN_PROGRESS.

Excepto para el entorno MTS (Microsoft Transaction Server), el valor predeterminado es MQCNO_HANDLE_SHARE_NONE. En el entorno MTS, el valor predeterminado es MQCNO_HANDLE_SHARE_BLOCK.

La llamada MQCONNX devuelve un descriptor de conexión. Las llamadas MQI posteriores de cualquier hebra del proceso pueden utilizar el descriptor, asociando dichas llamadas al descriptor devuelto por MQCONNX. Las llamadas MQI que utilicen un descritor compartido único se serializan en todas las hebras.

Por ejemplo, es posible realizar la siguiente secuencia de actividades con un descriptor compartido:

1. La hebra 1 emite MQCONNX y obtiene una manejador compartido *h1*
2. La hebra 1 abre una cola y emite una solicitud de obtención utilizando *h1*
3. La hebra 2 emite una solicitud de colocación utilizando *h1*
4. La hebra 3 emite una solicitud de colocación utilizando *h1*
5. La hebra 2 emite MQDISC utilizando *h1*

Mientras una hebra esté usando el descriptor, las demás hebras no tendrán acceso a la conexión. En aquellos casos en que sea aceptable que una hebra espere a que se complete cualquier llamada anterior de otra hebra, utilice MQCONNX con la opción MQCNO_HANDLE_SHARE_BLOCK.

No obstante, el bloqueo puede provocar dificultades. Suponga que en el paso “2” en la página 758 la hebra 1 emite una solicitud de obtención que espera a los mensajes que puedan no haber llegado aún

(una operación de obtención con espera). En este caso, las hebras 2 y 3 también se quedan a la espera (bloqueadas) mientras se lleva a cabo la solicitud de obtención en la hebra 1. Si prefiere que una llamada MQI devuelva un error si ya se está ejecutando otra llamada MQI sobre el descriptor de contexto, utilice MQCONNX con la opción MQCNO_HANDLE_SHARE_NO_BLOCK.

Notas de uso de una conexión compartida

1. Los descriptores de objeto (Hobj) creados al abrir un objeto se asocian a un Hconn; de ahí que, en el caso de un Hconn compartido, los Hobj también los comparta y pueda usar cualquier hebra que esté utilizando el Hconn. De forma similar, cualquier unidad de trabajo iniciada bajo Hconn se asocia a dicho Hconn; por lo que este también se comparte entre hebras con el Hconn compartido.
2. *Cualquier* hebra puede invocar MQDISC para desconectar un Hconn compartido, no solo la hebra que ha efectuado la llamada MQCONNX correspondiente. MQDISC finaliza el Hconn, lo que hace que no deje de estar disponible a ninguna hebra.
3. Una sola hebra puede utilizar varios Hconn compartidos de forma secuencial, por ejemplo, puede utilizar MQPUT para colocar un mensaje en un Hconn compartido y luego colocar otro mensaje utilizando otro Hconn compartido, realizándose cada operación bajo una unidad de trabajo local diferente.
4. Los Hconn compartidos no pueden utilizarse dentro de una unidad de trabajo global.

Multi *Uso de opciones de llamada MQCONNX con MQ_CONNECT_TYPE*

Utilice esta información para comprender las distintas opciones de llamada MQCONNX y cómo se utilizan con la variable de entorno **MQ_CONNECT_TYPE** .

Nota: MQ_CONNECT_TYPE sólo tiene efecto para los enlaces STANDARD. Para otros enlaces, se ignora **MQ_CONNECT_TYPE** .

En IBM MQ for Multiplatforms, puede utilizar la variable de entorno, **MQ_CONNECT_TYPE** en combinación con el tipo de enlace especificado en el campo Options de la estructura MQCNO utilizada en una llamada MQCONNX.

*Tabla 113. Cómo se utilizan las opciones de llamada MQCONNX con la variable de entorno **MQ_CONNECT_TYPE***

Opción de llamada de MQCONNX	Variable de entorno MQ_CONNECT_TYPE	Resultado
ESTÁNDAR	UNDEFINED	ESTÁNDAR
ESTÁNDAR	ESTÁNDAR	ESTÁNDAR
ESTÁNDAR	FASTPATH	ESTÁNDAR
ESTÁNDAR	CLIENTE	CLIENTE
ESTÁNDAR	LOCAL	ESTÁNDAR

Si no se especifica MQCNO_STANDARD_BINDING, puede utilizar MQCNO_NONE, que toma el valor predeterminado MQCNO_STANDARD_BINDING.

Autenticación e identidad para MQCONN y MQCONNX

Utilice esta tarea para aprender cómo las aplicaciones pueden proporcionar credenciales que se utilizan para la autenticación cuando se conectan a IBM MQ.

La identidad de usuario predeterminada

Cuando una aplicación utiliza la interfaz de cola de mensajes (MQI) para conectarse a IBM MQ con MQCONN o MQCONNX, siempre se establece una identidad de usuario y se asocia a la conexión.

De forma predeterminada, la identidad de usuario inicial es siempre la del proceso del sistema operativo bajo el que se ejecuta la aplicación. Esta identidad inicial puede ser suficiente para las conexiones de aplicación de confianza o enlazadas localmente.

Cuando una aplicación se conecta a un gestor de colas con una llamada MQCONN, la aplicación no puede modificar el ID de usuario predeterminado. Sin embargo, los mecanismos siguientes pueden cambiar el ID de usuario asociado a la conexión:

- Una salida de seguridad del lado del cliente o del lado del servidor.
- Reglas de autenticación de canal en el gestor de colas.
- Un ID de usuario de cliente establecido durante la autenticación mutua TLS.

Utilización de MQCONNX para proporcionar credenciales

MQCONNX proporciona a una aplicación más control sobre la identidad asociada con la conexión. Una aplicación puede proporcionar una estructura MQCSP como parte de las opciones de conexión especificadas en el parámetro **ConnectOpts** a MQCONNX. La estructura MQCSP puede contener credenciales que se utilizan para establecer una identidad de usuario. IBM MQ da soporte a las credenciales siguientes en la estructura MQCSP:

- Un ID de usuario y una contraseña.
- V 9.3.4 A partir de IBM MQ 9.3.4, una señal de autenticación, si la aplicación se conecta a un gestor de colas que se ejecuta en sistemas AIX o Linux .

La configuración de autenticación de conexión y autenticación de canal del gestor de colas controla cómo se procesan las credenciales proporcionadas por una aplicación. Por ejemplo, esta configuración afecta a los aspectos siguientes:

- Si las credenciales de la estructura MQCSP se validan y cómo se validan.
- Indica si el ID de usuario de las credenciales de la estructura MQCSP se correlaciona con otro ID de usuario.
- Indica si el usuario autenticado se adopta entonces como contexto para la aplicación.

Para obtener más información sobre la autenticación de conexión, consulte Autenticación de conexión. Para obtener más información sobre la autenticación de canal, consulte Registros de autenticación de canal.

Varios de los programas de ejemplo escritos en C que utilizan la MQI muestran cómo se utiliza la estructura MQCSP para proporcionar credenciales de autenticación. Para obtener más información, consulte los siguientes programas de ejemplo:

- “Programas de ejemplo de obtención” en la página 1119
- “Programas de transferencia de ejemplo” en la página 1132
- “El programa de ejemplo del navegador” en la página 1107
- “El programa de ejemplo TLS” en la página 1148

Información relacionada

Identificación y autenticación de usuarios utilizando la estructura MQCSP

MQCSP-Parámetros de seguridad

Identificación y autenticación de usuarios

Desconectar programas desde un gestor de colas utilizando MQDISC

Utilice esta información para aprender a desconectar programas desde un gestor de colas utilizando MQDISC.

Cuando un programa que se ha conectado a un gestor de colas utilizando la llamada MQCONN o MQCONNX ha finalizado todas las interacciones con el gestor de colas, interrumpe la conexión utilizando la llamada MQDISC, excepto:

- En las aplicaciones de CICS Transaction Server for z/OS donde la llamada es opcional, a menos que se haya utilizado MQCONNX y desee descartar el código de conexión antes de que finalice la aplicación.
- En IBM MQ for IBM i, donde se realiza una llamada MQDISC implícita cuando finaliza sesión en el sistema operativo.

Como entrada para la llamada MQDISC, debe proporcionar el descriptor de conexión (Hconn) que ha devuelto MQCONN o MQCONNX cuando se ha conectado al gestor de colas.

Excepto en CICS para z/OS, después de la llamada a MQDISC, el descriptor de conexión (Hconn) deja de ser válido y ya no puede realizar llamadas MQI adicionales hasta que vuelve a invocar MQCONN o MQCONNX. MQDISC no ejecuta MQCLOSE de forma implícita para cualquier objeto que continúe abierto utilizando este descriptor.

z/OS Para un cliente conectado a z/OS, cuando se emite una llamada MQDISC, se lleva a cabo una confirmación implícita, pero cualquier manejador de cola que continúe abierto no se cerrará hasta que realmente finalice el canal.

Si utiliza MQCONNX para la conexión en IBM MQ for z/OS, MQDISC también finaliza el ámbito del código de conexión establecido por MQCONNX. No obstante, en una aplicación CICS, IMS o RRS, si existe una unidad de recuperación activa asociada con un código de conexión, se rechaza MQDISC con un código de razón de MQRC_CONN_TAG_NOT_RELEASED.

Puede encontrar las descripciones de los parámetros en la sección MQDISC que describe la llamada MQDISC.

Cuando no se emite ninguna llamada MQDISC

Cuando se completa la creación de la hebra, se borra una conexión estándar y no compartida (Hconn). Una conexión compartida solo se restituye y desconecta de forma implícita cuando termina todo el proceso. Si finaliza la hebra que ha creado la conexión Hconn compartida, cuando todavía existe la conexión Hconn, esta conexión Hccon continúa siendo utilizable.

Comprobación de autorización

Normalmente, las llamadas MQCLOSE y MQDISC no comprueban la autorización.

En el desarrollo normal de una operación, un trabajo que tiene autorización para abrir o conectar con un objeto de IBM MQ cierra o se desconecta de dicho objeto. Incluso si se revoca la autorización de un trabajo que se ha conectado o ha abierto un objeto de IBM MQ, se aceptan las llamadas MQCLOSE y MQDISC.

Apertura y cierre de objetos

Esta información describe la apertura y cierre de objetos de IBM MQ.

Para realizar cualquiera de las operaciones siguientes, debe primero *abrir* el objeto pertinente de IBM MQ:

- Transferir un mensaje a una cola
- Obtener (examinar o recuperar) mensajes de una cola
- Establecer los atributos de un objeto
- Consultar los atributos de un objeto cualquiera

Utilice la llamada MQOPEN para abrir el objeto, utilizando las opciones de la llamada para especificar lo que desea hacer con el objeto. La única excepción es si desea colocar un mensaje en una cola y luego cerrar la cola inmediatamente. En este caso, puede pasar por alto la etapa de *apertura* utilizando la llamada MQPUT1 (consulte “Colocar un mensaje en una cola utilizando la llamada MQPUT1” en la página 781).

Antes de abrir un objeto utilizando la llamada MQOPEN, debe conectar su programa a un gestor de colas. Esto se explica con detalle, para todos los entornos, en "Conexión y desconexión de un gestor de colas" en la página 754.

Hay cuatro tipos de objeto de IBM MQ que puede abrir:

- Cola
- Lista de nombres
- Definición de proceso
- Gestor de colas

Puede abrir todos estos objetos de forma similar utilizando la llamada MQOPEN. Para obtener más información sobre los objetos de IBM MQ, consulte Tipos de objeto.

Puede abrir el mismo objeto más de una vez, y cada vez obtiene un nuevo descriptor de objeto. Puede desear examinar los mensajes de una cola utilizando un descriptor y eliminar mensajes de la misma cola utilizando otro descriptor. Esto ahorra la utilización de recursos para cerrar y reabrir el mismo objeto. También puede abrir una cola para examinar *y* eliminar mensajes al mismo tiempo.

Además, puede abrir varios objetos con una sola llamada MQOPEN y cerrarlos utilizando MQCLOSE. Consulte "Listas de distribución" en la página 783 para obtener información sobre cómo hacer esto.

Cuando un usuario intenta abrir un objeto, el gestor de colas comprueba que el usuario tenga autorización para abrir ese objeto para las opciones especificadas en la llamada MQOPEN.

Los objetos se cierran automáticamente cuando un programa se desconecta del gestor de colas. En el entorno IMS, la desconexión se fuerza cuando un programa inicia el proceso para un nuevo usuario después de una llamada GU (get unique) de IMS. En la plataforma IBM i, los objetos se cierran automáticamente cuando finaliza un trabajo.

Es una buena práctica de programación cerrar los objetos que ha abierto. Para ello utilice la llamada MQCLOSE.

Utilice los enlaces siguientes para obtener más información sobre la apertura y cierre de objetos:

- "Abrir objetos con la llamada MQOPEN" en la página 763
- "Creación de colas dinámicas" en la página 770
- "Apertura de colas remotas" en la página 771
- "Cierre de objetos utilizando la llamada MQCLOSE" en la página 771

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Colocación de mensajes en una cola" en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886

Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Abrir objetos con la llamada MQOPEN

Utilice esta información para obtener información acerca de cómo abrir objetos con la llamada MQOPEN.

Como entrada para la llamada MQOPEN, debe proporcionar lo siguiente:

- Un descriptor de conexión. En las aplicaciones CICS de z/OS, puede especificar la constante MQHC_DEF_HCONN (cuyo valor es cero), o utilizar el descriptor de conexión que ha devuelto la llamada MQCONN o MQCONNX. En el caso de otros programas, utilice el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX.
- Una descripción del objeto que desea abrir utilizando la estructura del descriptor de objeto (MQOD).
- Una o varias opciones que controlan la acción de la llamada.

La salida de MQOPEN es:

- Un descriptor de objeto que representa su acceso a dicho objeto. Utilice este descriptor para especificarlo en las llamadas MQI siguientes.
- Una estructura de descriptor de objeto modificada, si está creando una cola dinámica y ésta solo esta soportada en su plataforma.
- Un código de terminación.
- Un código de razón.

Ámbito de un descriptor de objeto

El ámbito de un descriptor de objeto (Hobj) es el mismo que el ámbito del descriptor de conexión (Hconn).

Esto se describe en la sección "Ámbito de MQCONN o MQCONNX" en la página 756 y la sección "Conexiones (independientes de hebra) compartidas con MQCONNX" en la página 758. No obstante, hay consideraciones adicionales para algunos entornos:

CICS
: En un programa CICS, solo puede utilizar el descriptor en la misma tarea de CICS desde la que realiza la llamada MQOPEN.

IMS y z/OS por lotes
: En los entornos IMS y por lotes, puede utilizar el descriptor en la misma tarea pero no en ninguna subtarea.

Puede encontrar las descripciones de los parámetros de la llamada MQOPEN en la sección MQOPEN.

Las secciones siguientes describen la información que debe proporcionar como entrada para MQOPEN.

Identificación de objetos (la estructura de MQOD)

Utilice la estructura de MQOD para identificar el objeto que desea abrir. Esta estructura es un parámetro de entrada para la llamada MQOPEN. El gestor de colas modifica la estructura cuando utiliza la llamada MQOPEN para crear una cola dinámica.

Para obtener información detallada acerca de la estructura de MQOD, consulte la sección MQOD.

Para obtener información acerca de cómo utilizar la estructura MQOD para las listas de distribución, consulte la sección "Uso de la estructura MQOD" en la página 784 bajo "Listas de distribución" en la página 783.

Resolución de nombres
Cómo la llamada MQOPEN resuelve los nombres de colas y de gestor de colas.

Nota: Un alias de gestor de colas es una definición de cola remota sin un campo RNAME.

Cuando abre una cola de IBM MQ, la llamada MQOPEN realiza una función de resolución de nombres en el nombre de cola que especifique. Esto determina en qué cola realiza las operaciones posteriores el gestor de colas. Esto significa que cuando especifica el nombre de una cola alias o de una cola remota en el descriptor de objetos (MQOD), la llamada resuelve el nombre en una cola local, o bien en una cola de transmisión. Si una cola se ha abierto para una entrada, un examen o para establecerla, se resuelve en una cola local, si existe alguna, y falla si no existe ninguna. Se resuelve en una cola local si se ha abierto sólo para salida, sólo para consultas, o sólo para salida y consultas. Consulte la Tabla 114 en la página 764 para obtener una visión general del proceso de resolución de nombres. El nombre que ha proporcionado en *ObjectQMgrName* se resuelve *antes* que en *ObjectName*.

En la Tabla 114 en la página 764 también se muestra cómo puede utilizar una definición local de una cola remota para definir un alias para el nombre de un gestor de colas. Esto le permite seleccionar qué cola de transmisión se utiliza cuando se colocan mensajes en una cola remota, de manera que puede, por ejemplo, utilizar una cola de transmisión única para los mensajes destinados a varios gestores de colas remotos.

Para poder utilizar la tabla siguiente, lea primero las dos columnas de la izquierda, que aparecen debajo de la cabecera **Entrada a MQOD**, y seleccione el caso pertinente. A continuación, lea la fila correspondiente, siguiendo las instrucciones pertinentes. Siguiendo las instrucciones de la columna **Nombres resueltos**, puede volver a las columnas **Entrada a MQOD** e insertar los valores tal como se indica, o bien puede salir de la tabla con los resultados proporcionados. Por ejemplo, es posible que se le solicite que lleve a cabo la entrada *ObjectName*.

Tabla 114. Resolución de nombres de cola cuando se utiliza MQOPEN

Entrada a MQOD	**Entrada a MQOD**	**Nombres resueltos**	**Nombres resueltos**	**Nombres resueltos**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
En blanco o gestor de colas local	Cola local sin el atributo CLUSTER	Gestor de colas local	Entrada *ObjectName*	No es aplicable (se está utilizando la cola local)
Gestor de colas en blanco	Cola local con el atributo CLUSTER	El gestor de colas de clúster seleccionado por la gestión de carga de trabajo o el gestor de colas de clúster específico seleccionado en PUT	Entrada *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE y cola local utilizada SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)
Gestor de colas local	Cola local con el atributo CLUSTER	Gestor de colas local	Entrada *ObjectName*	No es aplicable (se está utilizando la cola local)
En blanco o gestor de colas local	Cola modelo	Gestor de colas local	Nombre generado	No es aplicable (se está utilizando la cola local)

Tabla 114. Resolución de nombres de cola cuando se utiliza MQOPEN (continuación)

Entrada a MQOD	Entrada a MQOD	Nombres resueltos	Nombres resueltos	Nombres resueltos
En blanco o gestor de colas local	Cola alias con o sin el atributo CLUSTER	Vuelva a efectuar la resolución de nombres sin modificar *ObjectQMgrName*, y la entrada *ObjectName* establecida en *BaseQName* en el objeto de definición de cola alias. No se debe resolver en un alias definido localmente cuando se especifica la *ObjectQMgrName*, sino que se puede resolver en un alias en clúster (alojado en otros gestores de colas) donde *ObjectQMgrName* esté en blanco.		
Gestor de colas local	Cola alias con el atributo CLUSTER	El alias no se debe resolver en una cola de clúster que no se haya definido localmente, o una cola de clúster que tenga el mismo valor *ObjectName* que el alias.		
Gestor de colas en blanco	Cola alias con el atributo CLUSTER	El alias se puede resolver en una cola de clúster que tenga el mismo valor *ObjectName* que el alias.		
En blanco o gestor de colas local	Definición Local de una cola remota	Vuelva a efectuar la resolución de nombres con el valor *ObjectQMgrName* establecido en *RemoteQMgrName*, y el valor *ObjectName* establecido en *RemoteQName*. No se debe resolver en las colas remotas.		El nombre del atributo *XmitQName*, si no está en blanco; de lo contrario *RemoteQMgrName* en el objeto de definición de cola remota. SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)

Tabla 114. Resolución de nombres de cola cuando se utiliza MQOPEN (continuación)

Entrada a MQOD	**Entrada a MQOD**	**Nombres resueltos**	**Nombres resueltos**	**Nombres resueltos**
Gestor de colas en blanco	No hay ningún objeto local que coincida; se ha encontrado una cola de clúster	El gestor de colas de clúster seleccionado por la gestión de carga de trabajo o el gestor de colas de clúster específico seleccionado en PUT	Entrada *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)
En blanco o gestor de colas local	No hay ningún objeto local que coincida; no se ha encontrado ninguna cola de clúster		Error, no se ha encontrado ninguna cola	No aplicable
Nombre del gestor de colas en el mismo grupo de compartición de colas que el gestor de colas local	Cola local compartida	Gestor de colas local	Entrada *ObjectName*	No aplicable
El nombre de una cola de transmisión local	(No resuelto)	Entrada *ObjectQMgrName*	Entrada *ObjectName*	Entrada *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)
La definición de alias de gestor de colas (*RemoteQMgrName* puede ser el gestor de colas local)	(No resuelto, cola remota)	Vuelva a efectuar la resolución de nombres con el valor *ObjectQMgrName* establecido en el valor *RemoteQMgrName*. No se debe resolver en las colas remotas.	Entrada *ObjectName*	El nombre del atributo *XmitQName*, si no está en blanco; de lo contrario *RemoteQMgrName* en el objeto de definición de cola remota. SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)
El gestor de colas no es el nombre de ningún objeto local; se han encontrado gestores de colas de clúster o alias de gestor de colas.	(No resuelto)	*ObjectQMgrName* o el gestor de colas de clúster específico seleccionado en PUT	Entrada *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)
El gestor de colas no es el nombre de ningún objeto local; no se han encontrado objetos de clúster.	(No resuelto)	Entrada *ObjectQMgrName*	Entrada *ObjectName*	Atributo *DefXmitQName* del gestor de colas donde se da soporte a *DefXmitQName*. SYSTEM.QSG.TRANSMIT.QUEUE (véase la nota)

Notas:

1. *BaseQName* es el nombre de la cola base de la definición de la cola alias.

2. *RemoteQName* es el nombre de la cola remota de la definición local de la cola remota.
3. *RemoteQMgrName* es el nombre del gestor de colas remoto de la definición local de la cola remota.
4. *XmitQName* es el nombre de la cola de transmisión de la definición local de la cola remota.
5. Al utilizar gestores de colas IBM MQ for z/OS que forman parte de un grupo de compartición de colas (QSG), se puede utilizar el nombre del grupo de compartición de colas, en lugar del nombre del gestor de colas local en Tabla 114 en la página 764.

   Si el gestor de colas no puede abrir la cola de destino, o colocar un mensaje en la cola, el mensaje se transfiere al ObjectQMgrName especificado mediante la transferencia a colas dentro del grupo o mediante un canal IBM MQ.
6. En la columna *ObjectName* de la tabla, CLUSTER hace referencia tanto al atributo CLUSTER como al atributo CLUSNL de la cola.
7. Se utiliza SYSTEM.QSG.TRANSMIT.QUEUE si los gestores de colas locales y remotos están en el mismo grupo de compartición de colas; la colocación en colas entre grupos está habilitada.
8. Si ha asignado una cola de transmisión de clúster diferente a cada canal de clúster emisor, SYSTEM.CLUSTER.TRANSMIT.QUEUE no puede ser el nombre de la cola de transmisión de clúster. Para obtener más información acerca de varias colas de transmisión de clúster, consulte la sección Agrupación en clúster: Planificar cómo configurar las colas de transmisión de clúster .
9. Si el gestor de colas no es el nombre de ningún objeto local; se han encontrado gestores de colas de clúster o alias de gestor de colas.

   Cuando proporciona un nombre de gestor de colas utilizando **ObjectQMgrName** y hay varios canales de clúster con diferentes nombres de clúster, conocidos por el gestor de colas local, que pueden llegar a dicho destino, cualquiera de estos canales se puede utilizar para mover el mensaje, independientemente del nombre de clúster del destino.

   Esto puede no ser lo esperado, si tenía previsto que solo se enviasen los mensajes para dicha cola a través de un canal con el mismo nombre de clúster que la cola.

   No obstante, **ObjectQMgrName** tiene prioridad en este caso y el equilibrio de carga de trabajo del clúster tiene en cuenta todos los canales que pueden llegar a dicho gestor de colas, independientemente del nombre de clúster en el que se encuentran.

Al abrir una cola alias también se abre la cola base en la que se resuelve el alias, y al abrir una cola remota también se abre la cola de transmisión. Por tanto, no puede suprimir ni la cola que especifique ni la cola en la que se resuelve, mientras que la otra está abierta.

Mientras que una cola alias no se puede resolver en otra cola alias definida localmente (compartida en un clúster o no), se le permite resolver en una cola alias de clúster remota que se haya definido y, por tanto, puede especificarse como la cola base.

El nombre de cola resuelto y el nombre del gestor de colas resuelto se almacenan en los campos *ResolvedQName* y *ResolvedQMgrName*, del MQOD.

Para obtener más información sobre la resolución de nombres en un entorno de gestión de colas distribuidas, consulte ¿Qué es la resolución de nombres de cola?.

Utilización de las opciones de la llamada MQOPEN

En el parámetro **Options** de la llamada MQOPEN, debe elegir una o varias opciones para controlar el acceso que se le ha concedido para el objeto que va a abrir. Con estas opciones, puede:

- Abrir una cola y especificar que todos los mensajes transferidos a dicha cola se dirijan a la misma instancia de la misma
- Abrir una cola para poder colocar mensajes en la misma
- Abrir una cola para poder examinar mensajes de la misma
- Abrir una cola para poder eliminar mensajes de la misma
- Abrir un objeto para poder consultar y establecer sus atributos, aunque solo puede establecer los atributos de las colas

- Abrir un tema o serie de tema para publicar mensajes
- Asociar información de contexto a un mensaje
- Designar un identificador de usuario alternativo para utilizarlo en las comprobaciones de seguridad
- Controlar la llamada si el gestor de colas está en estado de desactivación temporal

Opción MQOPEN para cola de clúster
El enlace utilizado para el manejador de cola se toma del atributo de cola **DefBind**, que puede tomar el valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, o MQBND_BIND_ON_GROUP.

Para direccionar todos los mensajes colocados en una cola utilizando MQPUT al mismo gestor de colas por la misma ruta, utilice la opción MQOO_BIND_ON_OPEN en la llamada MQOPEN.

Para especificar que se va a seleccionar un destino en el momento de MQPUT, es decir, en base mensaje-a-mensaje, utilice la opción MQOO_BIND_NOT_FIXED en la llamada MQOPEN.

Para especificar que todos los mensajes de un put de grupo de mensajes a una cola utilizando MQPUT se asignen a la misma instancia de destino, utilice la opción MQOO_BIND_ON_GROUP en la llamada MQOPEN.

Se debe especificar MQOO_BIND_ON_OPEN o MQOO_BIND_ON_GROUP cuando se utilizan grupos de mensajes con clústeres para asegurarse de que todos los mensajes del grupo se procesan en el mismo destino.

Si no especifica ninguna de estas opciones, se utiliza el valor predeterminado, MQOO_BIND_AS_Q_DEF.

Si especifica el nombre de un gestor de colas en el MQOD, la cola de ese gestor de colas está seleccionada. Si el nombre del gestor de colas está en blanco, se puede seleccionar cualquier instancia. Consulte "La llamada MQOPEN y los clústeres" en la página 907 para obtener más información.

Si abre una cola de clúster utilizando una definición QALIAS, algunos atributos de cola se definen mediante la cola de alias, y no la cola base. Los atributos de clúster se encuentran entre los atributos de la definición de la cola base que la cola alias ha alterado temporalmente. Por ejemplo, en el fragmento de código siguiente, la cola de clúster se abre con MQOO_BIND_NOT_FIXED y no MQOO_BIND_ON_OPEN. La definición de cola de clúster se anuncia en todo el clúster; la definición de la cola alias es local para el gestor de colas.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

Opción MQOPEN para colocar mensajes
Para abrir una cola o un tema para colocarles mensajes, utilice la opción MQOO_OUTPUT.

Opción MQOPEN para examinar mensajes
Para abrir una cola de forma que se puedan *examinar* sus mensajes, se usa la llamada MQOPEN con la opción MQOO_BROWSE.

Esto crea un *cursor para examinar* que utiliza el gestor de colas para identificar el siguiente mensaje en la cola. Para obtener más información, consulte "Cómo examinar mensajes en una cola" en la página 823.

Nota:

1. No se pueden examinar mensajes en una cola remota; no abra una cola remota utilizando la opción MQOO_BROWSE.
2. No puede especificar esta opción al abrir una lista de distribución. Para obtener más información sobre las listas de distribución, consulte "Listas de distribución" en la página 783.
3. Utilice MQOO_CO_OP junto con MQOO_BROWSE si está utilizando un examen cooperativo; consulte Opciones

Opciones de MQOPEN para eliminar mensajes
Tres opciones controlan la apertura de una cola para eliminar sus mensajes.

Solo puede utilizar uno de ellos en cualquier llamada MQOPEN. Estas opciones definen si el programa tiene acceso exclusivo o compartido a la cola. El *acceso exclusivo* significa que, hasta que cierre la cola, solo su programa puede eliminar los mensajes de la cola. Si otro programa intenta abrir la cola para eliminar mensajes, su llamada MQOPEN falla. El *acceso compartido* significa que más de un programa puede eliminar mensajes de la cola.

El método más aconsejable es aceptar el tipo de acceso que se pretendía para la cola cuando se definió. La definición de cola implicaba el establecimiento de los atributos **Shareability** y **DefInputOpenOption**. Para aceptar este acceso, utilice la opción MQOO_INPUT_AS_Q_DEF. Consulte la Tabla 115 en la página 769 para ver cómo afecta el valor de estos atributos al tipo de acceso que se le otorgará cuando utilice esta opción.

Tabla 115. Cómo afectan los atributos de cola y las opciones de la llamada MQOPEN al acceso a las colas

Atributos de colas		**Tipo de acceso con las opciones de MQOPEN**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	compartido	compartido	exclusivo
SHAREABLE	EXCLUSIVE	exclusivo	compartido	exclusivo
NOT_SHAREABLE*	SHARED*	exclusivo	exclusivo	exclusivo
NOT_SHAREABLE	EXCLUSIVE	exclusivo	exclusivo	exclusivo

Nota: * Aunque puede definir una cola para que tenga esta combinación de atributos, la opción de apertura de entrada predeterminada se altera temporalmente por el atributo shareability.

De forma alternativa:

- Si sabe que la aplicación puede funcionar correctamente aunque otros programas puedan eliminar mensajes de la cola al mismo tiempo, utilice la opción MQOO_INPUT_SHARED. La Tabla 115 en la página 769 muestra cómo, en algunos casos, tendrá acceso exclusivo a la cola, incluso con esta opción.
- Si sabe que la aplicación solo puede funcionar correctamente si otros programas no pueden eliminar mensajes de la cola al mismo tiempo, utilice la opción MQOO_INPUT_EXCLUSIVE.

Nota:

1. No puede eliminar mensajes de una cola remota. Por lo tanto, no puede abrir una cola remota utilizando ninguna de las opciones de MQOO_INPUT_*.
2. No puede especificar esta opción al abrir una lista de distribución. Para obtener más información, consulte “Listas de distribución” en la página 783.

Opciones de MQOPEN para establecer y consultar atributos
Para abrir una cola a fin de poder establecer sus atributos, utilice la opción MQOO_SET.

No se pueden establecer los atributos de ningún otro tipo de objeto (consulte “Consulta y establecimiento de atributos de objeto” en la página 871).

Para abrir un objeto a fin de poder consultar sus atributos, utilice la opción MQOO_INQUIRE.

Nota: No puede especificar esta opción al abrir una lista de distribución.

Opciones de MQOPEN relacionadas con el contexto del mensaje
Si desea poder asociar información de contexto a un mensaje cuando lo coloca en una cola, debe utilizar una de las opciones de contexto de mensaje al abrir la cola.

Las opciones permiten diferenciar entre la información de contexto que está relacionada con el *usuario* que ha originado el mensaje y la que está relacionada con la *aplicación* que ha originado el mensaje. Además, puede optar por establecer la información de contexto cuando coloca el mensaje en la cola, o puede optar por obtener el contexto automáticamente desde otro manejador de colas.

Conceptos relacionados

"Contexto de mensaje" en la página 48
La información del *contexto de mensaje* permite a la aplicación que recupera el mensaje averiguar quién ha originado el mensaje.

"Control de la información de contexto de mensaje" en la página 779
Cuando utilice la llamada MQPUT o MQPUT1 para colocar un mensaje en una cola, puede especificar que el gestor de colas añada alguna información de contexto predeterminada en el descriptor de mensaje. Las aplicaciones que tengan el nivel apropiado de autorización pueden añadir información de contexto adicional. Puede utilizar el campo de opciones de la estructura MQPMO para controlar la información de contexto.

Opción MQOPEN para autorización de usuario alternativa
Cuando se intenta abrir un objeto con la llamada MQOPEN, el gestor de colas comprueba si usted tiene autorización para abrir dicho objeto. Si no está autorizado, la llamada falla.

Sin embargo, puede que los programas de servidor deseen que el gestor de colas compruebe la autorización del usuario para el que están trabajando en lugar de la propia autorización del servidor. Para ello, deben utilizar la opción MQOO_ALTERNATE_USER_AUTHORITY de la llamada MQOPEN y especificar el ID de usuario alternativo en el campo *AlternateUserId* de la estructura MQOD. Por lo general, el servidor obtendría el ID de usuario de la información de contexto en el mensaje que está procesando.

z/OS *Opción MQOPEN para la desactivación temporal del gestor de colas*
Si se usa la llamada MQOPEN cuando un gestor de colas se encuentra en estado de desactivación temporal, dicha llamada podría fallar dependiendo del entorno que se esté usando.

En el entorno CICS en z/OS, si utiliza la llamada MQOPEN cuando el gestor de colas está en un estado de desactivación temporal, la llamada siempre falla.

En otros entornos z/OS y Multiplatforms, la llamada falla cuando el gestor de colas se desactiva temporalmente sólo si utiliza la opción MQOO_FAIL_IF_QUIESCING de la llamada MQOPEN.

Opción MQOPEN para resolver nombres de colas locales
Cuando abre una cola local, alias o modelo, se devuelve la cola local.

No obstante, cuando abre una cola remota o cola de clúster, los campos *ResolvedQName* y *ResolvedQMgrName* de la estructura MQOD se rellenan con los nombres de la cola remota y el gestor de colas remoto que se encuentran en la definición de la cola remota, o con la cola de clúster remoto elegido.

Utilice la opción MQOO_RESOLVE_LOCAL_Q de la llamada MQOPEN para rellenar *ResolvedQName* en la estructura MQOD con el nombre de la cola local que se ha abierto. El campo *ResolvedQMgrName* se rellena del mismo modo con el nombre del gestor de colas local que aloja la cola local. Este campo solo está disponible con la Versión 3 de la estructura MQOD. Si la versión de la estructura es inferior a la Versión 3, se omite MQOO_RESOLVE_LOCAL_Q y no se devuelve un error.

Si especifica MQOO_RESOLVE_LOCAL_Q al abrir, por ejemplo una cola remota, *ResolvedQName* es el nombre de la cola de transmisión en la que se colocarán los mensajes. El campo *ResolvedQMgrName* es el nombre del gestor de colas local que aloja la cola de transmisión.

Creación de colas dinámicas
Utilice una cola dinámica cuando no necesite la cola después de que finalice la aplicación.

Por ejemplo, puede utilizar una cola dinámica para la cola de respuestas. Especifique el nombre de la cola de respuesta en el campo *ReplyToQ* de la estructura MQMD cuando coloque un mensaje en una cola (consulte "Definición de mensajes utilizando la estructura MQMD" en la página 774).

Para crear una cola dinámica, utilice una plantilla conocida como cola de modelo, junto con la llamada MQOPEN. La cola de modelo se crea utilizando los mandatos de IBM MQ o las operaciones y los paneles de control. La cola dinámica que va a crear utiliza los atributos de la cola de modelo.

Cuando llame a MQOPEN, especifique el nombre de la cola de modelo en el campo *ObjectName* de la estructura MQOD. Cuando se completa la llamada, el campo *ObjectName* se establece en el nombre de la cola dinámica que se crea. Asimismo, el campo *ObjectQMgrName* se establece en el nombre del gestor de colas local.

Puede especificar el nombre de la cola dinámica que se crea de tres maneras:

- Proporcione el nombre completo que desee en el campo *DynamicQName* de la estructura MQOD.
- Especifique un prefijo (de menos de 33 caracteres) para el nombre y permita que el gestor de colas genere el resto del nombre. El gestor de colas genera un nombre exclusivo, pero usted todavía tienen algún control (por ejemplo, es posible que desee que cada usuario utilice un prefijo determinado, o que desee otorgar una clasificación de seguridad especial a las colas con un determinado prefijo en su nombre). Para utilizar este método, especifique un asterisco (*) para el último carácter que no está en blanco del campo *DynamicQName*. No especifique un asterisco (*) individual como nombre de cola dinámica.
- Permita que el gestor de colas genere el nombre completo. Para utilizar este método, especifique un asterisco (*) en la primera posición de carácter del campo *DynamicQName*.

Para obtener más información sobre estos métodos, consulte la descripción del campo DynamicQName.

Hay más información sobre las colas dinámicas en Colas dinámicas y de modelo.

Apertura de colas remotas

Una cola remota es una cola propiedad de un gestor de colas distinto al gestor con el que está conectada la aplicación.

Para abrir una cola remota, utilice la llamada MQOPEN del mismo modo que para una cola local. Puede especificar el nombre de la cola de la manera siguiente:

1. En el campo *ObjectName* de la estructura MQOD, especifique el nombre de la cola remota tal como lo conoce el gestor de colas *local* .

   Nota: Deje en blanco el campo *ObjectQMgrName* en este caso.
2. En el campo *ObjectName* de la estructura MQOD, especifique el nombre de la cola remota, tal como la conoce el gestor de colas *remoto*. En el campo *ObjectQMgrName*, especifique uno de los valores siguientes:

   - El nombre de cola de transmisión con el mismo nombre que el gestor de colas remoto. El nombre y las mayúsculas, minúsculas o la combinación de las mismas, debe coincidir *exactamente*.
   - El nombre de un objeto de alias de gestor de colas que se resuelve en el gestor de colas de destino en la cola de transmisión.

   Esto indica al gestor de colas el destino del mensaje, así como la cola de transmisión a la que debe transferirse para su recepción.
3. Si se da soporte a *DefXmitQname*, en el campo *ObjectName* de la estructura MQOD, especifique el nombre de la cola remota tal como la conoce el gestor de colas *remoto*.

   Nota: Establezca el campo *ObjectQMgrName* en el nombre del gestor de colas remoto (en este caso, no se puede dejar en blanco).

Solo los nombres locales se validan cuando invoca MQOPEN. La última comprobación es para comprobar que existe la cola de transmisión que se ha de utilizar.

Estos métodos se resumen en la Tabla 114 en la página 764.

Cierre de objetos utilizando la llamada MQCLOSE

Para cerrar un objeto, utilice la llamada MQCLOSE.

Si el objeto es una cola, tenga en cuenta lo siguiente:

- No es necesario vaciar una cola dinámica temporal antes de cerrarla.

Cuando se cierra una cola dinámica temporal, la cola se suprime, junto con los mensajes que pueda haber en ella. Esto es cierto incluso si hay llamadas MQGET, MQPUT o MQPUT1 no confirmadas pendientes en la cola.

- En IBM MQ for z/OS, si tiene alguna solicitud MQGET con una opción MQGMO_SET_SIGNAL pendiente para esa cola, se cancelará.
- Si ha abierto la cola utilizando la opción MQOO_BROWSE, el cursor para examinar se destruye.

El cierre no está relacionado con el punto de sincronización, por lo tanto, puede cerrar las colas antes o después del punto de sincronización.

Como entrada a la llamada MQCLOSE, debe proporcionar:

- Un descriptor de conexión. Utilice el mismo manejador de conexión que se ha utilizado para abrirlo o, de manera alternativa, para las aplicaciones CICS en z/OS, puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero).
- El manejador del objeto que desea cerrar. Obténgalo de la salida de la llamada MQOPEN.
- MQCO_NONE en el campo *Options* (a menos que esté cerrando una cola dinámica permanente).
- La opción de control para determinar si el gestor de colas debe suprimir la cola aunque todavía tenga mensajes (cuando se cierra una cola dinámica permanente).

La salida de MQCLOSE es:

- Un código de terminación
- Un código de razón
- El manejador de objetos, restablecido en el valor MQHO_UNUSABLE_HOBJ

Puede encontrar una descripción de los parámetros de la llamada MQCLOSE en MQCLOSE.

Colocación de mensajes en una cola

Utilice esta información para conocer cómo poner mensajes en una cola.

Utilice la llamada MQPUT para poner mensajes en una cola. Puede utilizar MQPUT repetidamente para poner muchos mensajes en la misma cola, a continuación de la llamada MQOPEN inicial. Invoque MQCLOSE cuando haya terminado de colocar todos los mensajes en la cola.

Si desea poner un solo mensaje en una cola y cerrar la cola inmediatamente después, puede utilizar la llamada MQPUT1. MQPUT1 realiza las mismas funciones que la siguiente secuencia de llamadas:

- MQOPEN
- MQPUT
- MQCLOSE

Sin embargo, por lo general, si tiene más de un mensaje para poner en la cola, es más eficaz utilizar la llamada MQPUT. Esto depende del tamaño del mensaje y de la plataforma en la que está trabajando.

Utilice los enlaces siguientes para obtener más información sobre la colocación de mensajes en una cola:

- “Transferencia de mensajes a una cola local utilizando la llamada MQPUT” en la página 773
- “Colocación de mensajes en una cola remota” en la página 778
- “Establecimiento de las propiedades de un mensaje” en la página 778
- “Control de la información de contexto de mensaje” en la página 779
- “Colocar un mensaje en una cola utilizando la llamada MQPUT1” en la página 781
- “Listas de distribución” en la página 783
- “Algunos casos en los que las llamadas put fallan” en la página 787

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Apertura y cierre de objetos" en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

"Obtención de mensajes de una cola" en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Transferencia de mensajes a una cola local utilizando la llamada MQPUT

Utilice esta información para obtener información sobre cómo poner mensajes en una cola local utilizando la llamada MQPUT.

Como entrada para la llamada MQPUT, debe proporcionar lo siguiente:

- Un manejador de conexión (Hconn).
- Un manejador de cola (Hobj).
- Una descripción del mensaje que desea colocar en la cola. Tiene el formato de una estructura de descriptor de mensaje (MQMD).
- La información de control en formato de una estructura de opciones de transferencia de mensaje (MQPMO).
- La longitud de los datos contenidos dentro del mensaje (MQLONG).
- Los propios datos del mensaje.

La salida de la llamada MQPUT es la siguiente:

- Un código de razón (MQLONG)
- Un código de terminación (MQLONG)

Si la llamada se completa de forma satisfactoria, también devuelve la estructura de las opciones y la estructura del descriptor de mensaje. La llamada modifica la estructura de las opciones para mostrar el nombre de la cola y el gestor de colas donde se envió el mensaje. Si solicita que el gestor de colas genere un valor exclusivo para el identificador del mensaje que está transfiriendo (especificando un cero binario

en el campo *MsgId* de la estructura MQMD), la llamada inserta el valor en el campo *MsgId* antes de devolverle esta estructura. Restablezca este valor antes de emitir otra llamada MQPUT.

Hay una descripción de la llamada MQPUT en MQPUT.

Para obtener una descripción más detallada de la información necesaria para la entrada de la llamada MQPUT, consulte los enlaces siguientes:

- "Especificación de manejadores" en la página 774
- "Definición de mensajes utilizando la estructura MQMD" en la página 774
- "Especificación de opciones utilizando la estructura MQPMO" en la página 774
- "Los datos del mensaje" en la página 777
- "Transferencia de mensajes: uso de manejadores de mensaje" en la página 778

Especificación de manejadores

Para el manejador de conexión (*Hconn*) en CICS en las aplicaciones z/OS, puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero), o puede utilizar el manejador de conexión devuelto por la llamada MQCONN o MQCONNX. Para otras aplicaciones, utilice siempre el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX.

Sea cual sea el entorno en el que trabaje, utilice el mismo manejador de cola (*Hobj*) que el que devuelve la llamada MQOPEN.

Definición de mensajes utilizando la estructura MQMD

La estructura de descriptor de mensaje (MQMD) es un parámetro de entrada/salida para las llamadas MQPUT y MQPUT1. Utilícela para definir el mensaje que está poniendo en la cola.

Si se especifica MQPRI_PRIORITY_AS_Q_DEF o MQPER_PERSISTENCE_AS_Q_DEF para el mensaje y la cola es una cola de clúster, los valores utilizados son aquellos de la cola en los que se resuelve MQPUT. Si dicha cola está inhabilitada para MQPUT, la llamada fallará. Consulte Configuración de un clúster de gestor de colas para obtener más información.

Nota: Utilice MQPMO_NEW_MSG_ID y MQPMO_NEW_CORREL_ID antes de poner un nuevo mensaje para garantizar que *MsgId* y *CorrelId* sean exclusivos. Los valores de estos campos se devuelven en una llamada MQPUT realizada con éxito.

Hay una introducción a las propiedades de mensaje que MQMD describe en "Mensajes de IBM MQ" en la página 18, y existe una descripción de la propia estructura en MQMD.

Especificación de opciones utilizando la estructura MQPMO

Utilice la estructura MQPMO (opciones de transferencia de mensajes) para pasar opciones a las llamadas MQPUT y MQPUT1.

En las secciones siguientes se proporciona ayuda para rellenar los campos de esta estructura. Hay una descripción de la estructura en MQPMO.

La estructura incluye los campos siguientes:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*

- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

El contenido de estos campos es el siguiente:

StrucId
Esto identifica la estructura como una estructura de opciones de transferencia de mensajes. Se trata de un campo de 4 caracteres. Especifique siempre MQPMO_STRUC_ID.

Versión
Describe el número de versión de la estructura. El valor predeterminado es MQPMO_VERSION_1. Si especifica MQPMO_VERSION_2, puede utilizar listas de distribución (consulte "Listas de distribución" en la página 783). Si especifica MQPMO_VERSION_3, puede utilizar manejadores de mensajes y propiedades de mensajes. Si especifica MQPMO_CURRENT_VERSION, su aplicación siempre está configurada para utilizar el nivel más reciente.

Opciones
Controla lo siguiente:

- Si la operación de transferencia está o no incluida en una unidad de trabajo
- Cuánta información de contexto hay asociada a un mensaje
- De dónde procede la información de contexto
- Si la llamada falla si el gestor de colas está en estado de desactivación temporal
- Si se permite o no el agrupamiento o la segmentación
- Generación de un nuevo identificador de mensaje y un identificador de correlación
- El orden en que los mensajes y segmentos se ponen en una cola
- Si se resuelven o no los nombres de cola locales

Si deja el campo *Options* establecido en el valor predeterminado (MQPMO_NONE), el mensaje que coloque tendrá asociada la información de contexto predeterminada.

Además, el modo en que funciona la llamada con puntos de sincronización lo determina la plataforma. El valor predeterminado del control de punto de sincronización es yes (sí) en z/OS; para otras plataformas, es no.

Contexto
Indica el nombre del manejador de cola desde el que desea que se copie la información de contexto (si se solicita en el campo *Options*).

Para ver una introducción al contexto de mensaje, consulte "Contexto de mensaje" en la página 48. Para obtener información sobre el uso de la estructura MQPMO para controlar la información de contexto de un mensaje, consulte "Control de la información de contexto de mensaje" en la página 779.

ResolvedQName
Contiene el nombre (tras la resolución de cualquier nombre de alias) de la cola que se ha abierto para recibir el mensaje. Se trata de un campo de salida.

ResolvedQMgrName
Contiene el nombre (tras la resolución de cualquier nombre de alias) del gestor de colas propietario de la cola en *ResolvedQName*. Se trata de un campo de salida.

MQPMO también puede dar cabida a los campos necesarios para las listas de distribución ("Listas de distribución" en la página 783). Si desea utilizar este recurso, se utiliza la versión 2 de la estructura MQPMO. Incluye los campos siguientes:

RecsPresent
Este campo contiene el número de colas de la lista de distribución; es decir, el número de registros de transferencia de mensajes (MQPMR) y los correspondientes registros de respuesta (MQRR) presentes.

El valor que especifique puede ser el mismo que el número de registros de objetos proporcionado en MQOPEN. No obstante, si el valor es menor que el número de registros de objetos proporcionados en la llamada MQOPEN, o si no proporciona registros de transferencia de mensaje, los valores de las colas que no estén definidos se toman de los valores predeterminados proporcionados por el descriptor de mensaje. Además, si el valor es mayor que el número de registros de objetos proporcionado, se pasarán por alto los registros de transferencia de mensajes sobrantes.

Se recomienda que realice una de las acciones siguientes:

- Si desea recibir un informe o una respuesta de cada destino, especifique el mismo valor que el que aparece en la estructura MQOR y utilice MQPMR que contengan campos *MsgId*. Inicialice estos campos *MsgId* con ceros o especifique MQPMO_NEW_MSG_ID.

  Cuando haya transferido el mensaje a la cola, los valores de *MsgId* que haya creado el gestor de colas pasarán a estar disponibles en los MQPMR; puede utilizarlos para identificar qué destino está asociado con cada informe o respuesta.
- Si no desea recibir informes ni respuestas, elija una de las opciones siguientes:
  1. Si desea identificar destinos que fallan de forma inmediata, es posible que todavía desee especificar el mismo valor en el campo *RecsPresent* que el aparece en la estructura MQOR y proporcionar MQRR para identificar dichos destinos. No especifique ningún MQPMR.
  2. Si no desea identificar destinos con error, especifique cero en el campo *RecsPresent* y no proporcione MQPMR ni MQRR.

Nota: Si utiliza MQPUT1, el número de punteros de registro de respuesta y de desplazamientos de registro de respuesta debe ser cero.

Para ver una descripción completa de los registros de transferencia de mensajes (MQPMR) y los registros de respuesta (MQRR), consulte MQPMR y MQRR.

PutMsgRecFields
Indica qué campos están presentes en cada registro de transferencia de mensajes (MQPMR). Para obtener una lista de estos campos, consulte "Uso de la estructura MQPMR" en la página 786.

PutMsgRecOffset* y *PutMsgRecPtr
Los punteros (normalmente en C) y los desplazamientos (normalmente en COBOL) se utilizan para direccionar los registros de transferencia de mensajes (consulte "Uso de la estructura MQPMR" en la página 786 para obtener una visión general de la estructura MQPMR).

Utilice el campo *PutMsgRecPtr* para especificar un puntero al primero registro de transferencia de mensajes, o el campo *PutMsgRecOffset* para especificar el desplazamiento del primero registro de transferencia de mensajes. Este es el desplazamiento desde el inicio del MQPMO. En función del campo *PutMsgRecFields*, especifique un valor no nulo para *PutMsgRecOffset* o *PutMsgRecPtr*.

ResponseRecOffset y ResponseRecPtr
También puede utilizar punteros y desplazamientos para direccionar los registros de respuesta (consulte "Uso de la estructura MQRR" en la página 785 para obtener más información sobre los registros de respuesta).

Utilice el campo *ResponseRecPtr* para especificar un puntero al primer registro de respuesta, o el campo *ResponseRecOffset* para especificar el desplazamiento del primero registro de respuesta. Este es el desplazamiento desde el inicio de la estructura MQPMO. Especifique un valor no nulo para *ResponseRecOffset* o *ResponseRecPtr*.

Nota: Si utiliza MQPUT1 para transferir mensajes a una lista de distribución, *ResponseRecPtr* debe ser nulo o cero y *ResponseRecOffset* debe ser cero.

La versión 3 de la estructura MQPMO incluye también los campos siguientes:

OriginalMsgHandle
El uso que puede hacer de este campo depende del valor del campo *Action*. Si va a transferir un mensaje nuevo con propiedades de mensaje asociadas, establezca este campo en el manejador de mensaje que haya creado anteriormente y sobre el cual haya establecido las propiedades. Si está reenviando, respondiendo o generando un informe en respuesta a un mensaje recuperado anteriormente, este campo contiene el manejador de mensaje de dicho mensaje.

NewMsgHandle
Si especifica un *NewMsgHandle*, las propiedades asociadas con el manejador alteran temporalmente las propiedades asociadas con el *OriginalMsgHandle*. Para obtener más información, consulte Action (MQLONG).

Acción
Utilice este campo para especificar el tipo de transferencia que se realiza. Los valores posibles y sus significados son los siguientes:

MQACTP_NEW
Nuevo mensaje sin relación con ningún otro.

MQACTP_FORWARD
Mensaje recuperado anteriormente y que se está reenviando ahora.

MQACTP_REPLY
Mensaje de respuesta a un mensaje recuperado con anterioridad.

MQACTP_REPORT
Mensaje que es un informe generado como resultado de un mensaje recuperado con anterioridad.

Para obtener más información, consulte Action (MQLONG).

PubLevel
Si este mensaje es una publicación, puede establecer este campo para determinar qué suscripciones recibe. Solo recibirán esta publicación las suscripciones que tengan un *SubLevel* menor o igual a este valor. El valor predeterminado es 9, que es el nivel más alto e implica que las suscripciones con cualquier *SubLevel* pueden recibir esta publicación.

Los datos del mensaje

Proporcione la dirección del almacenamiento intermedio que contiene los datos en el parámetro **Buffer** de la llamada MQPUT. Puede incluir cualquier cosa en los datos de sus mensajes. La cantidad de datos de los mensajes, no obstante, afectará al rendimiento de la aplicación que los procesa.

El tamaño máximo de los datos viene determinado por:

- El atributo **MaxMsgLength** del gestor de colas
- El atributo **MaxMsgLength** de la cola a la que transfiere el mensaje
- El tamaño de cualquier cabecera de mensaje añadida por IBM MQ (incluyendo la cabecera de mensaje no entregado, MQDLH, y la cabecera de lista de distribución, MQDH)

El atributo **MaxMsgLength** del gestor de colas contiene el tamaño de mensaje que el gestor de colas puede procesar. Este tiene un valor predeterminado de 100 MB para todos los productos de IBM MQ con V6 o superior.

Para determinar el valor de este atributo, utilice la llamada MQINQ sobre el objeto de gestor de colas. Para mensajes grandes, puede cambiar este valor.

El atributo **MaxMsgLength** de una cola determina el tamaño máximo de mensaje que se puede transferir a la cola. Si intenta transferir un mensaje con un tamaño mayor que el valor de este atributo, la llamada MQPUT fallará. Si transfiere un mensaje a una cola remota, el tamaño máximo de mensaje que puede transferir correctamente viene determinado por el atributo **MaxMsgLength** de la cola remota, de las colas de transmisión intermedias a las que se transfiere el mensaje a lo largo del camino hacia su destino, y de los canales utilizados.

Para una operación MQPUT, el tamaño del mensaje debe ser menor o igual que el atributo **MaxMsgLength** tanto de la cola como del gestor de colas. Los valores de estos atributos son

independientes, pero se recomienda que establezca el atributo *MaxMsgLength* de la cola en un valor menor o igual que el del gestor de colas.

IBM MQ añade información de cabecera a los mensajes en las circunstancias siguientes:

- Al transferir un mensaje a una cola remota, IBM MQ añade una estructura de cabecera de transmisión (MQXQH) al mensaje. Esta estructura incluye el nombre de la cola de destino y su propio gestor de colas.
- Si IBM MQ no puede entregar un mensaje a una cola remota, intenta transferir el mensaje a la cola de mensajes no entregados. Añade una estructura MQDLH al mensaje. Esta estructura incluye el nombre de la cola de destino y la razón por la que se ha transferido el mensaje a la cola de mensajes no entregados.
- Si desea enviar un mensaje a varias colas de destino, IBM MQ añadirá una cabecera MQDH al mensaje. Esta cabecera describe los datos presentes en un mensaje, perteneciente a una lista de distribución, en una cola de transmisión. Tenga en cuenta esto al elegir un valor óptimo para la longitud máxima de mensaje.
- Si el mensaje es un segmento o un mensaje de un grupo, es posible que IBM MQ añada un MQMDE.

Estas estructuras se describen en MQDH y MQMDE.

Si sus mensajes tienen el tamaño máximo permitido para estas colas, la adición de estas cabeceras implica que las operaciones de transferencia fallarán debido a que los mensajes son ahora demasiado grandes. Para reducir la posibilidad de que fallen las operaciones de transferencia:

- Haga que el tamaño de sus mensajes sea menor que el atributo **MaxMsgLength** de las colas de transmisión y de mensajes no entregados. Permita al menos el valor de la constante MQ_MSG_HEADER_LENGTH (más para listas de distribución grandes).
- Asegúrese de que el atributo **MaxMsgLength** de la cola de mensajes no entregados esté establecido en el mismo valor que el atributo *MaxMsgLength* del gestor de colas propietario de la cola de mensajes no entregados.

Los atributos del gestor de colas y las constantes de colas de mensajes se describen en Atributos del gestor de colas.

z/OS Para obtener información sobre cómo se gestionan los mensajes no entregados en un entorno de colas distribuidas, consulte Mensajes no entregados y no procesados.

Transferencia de mensajes: uso de manejadores de mensaje

Hay dos manejadores de mensaje disponibles en la estructura MQPMO, *OriginalMsgHandle* y *NewMsgHandle*. La relación entre estos manejadores de mensaje está definida por el campo *Action* de MQPMO.

Para obtener detalles completos, consulte Action (MQLONG). Un manejador de mensaje no es necesariamente obligatorio para transferir un mensaje. Su finalidad es asociar propiedades con un mensaje, por lo que solo es necesario si se utilizan propiedades de mensaje.

Colocación de mensajes en una cola remota

Cuando desea poner un mensaje en una cola remota (es decir, una cola que es propiedad de un gestor de colas distinto de aquel al está conectada la aplicación) en lugar de una cola local, la única consideración adicional es cómo especifica el nombre de la cola cuando la abre.Esto se describe en “Apertura de colas remotas” en la página 771. No hay ningún cambio respecto a cómo utiliza la llamada MQPUT o MQPUT1 para una cola local.

Para obtener más información sobre cómo utilizar colas remotas y colas de transmisión, consulte Técnicas de colocación en colas distribuidas de IBM MQ.

Establecimiento de las propiedades de un mensaje

Llame a MQSETMP para cada propiedad que desee establecer. Al transferir el mensaje, establezca el manejador de mensajes y los campos de acción de la estructura MQPMO.

Para asociar propiedades con un mensaje, el mensaje debe tener un manejador de mensajes. Cree un manejador de mensajes utilizando la llamada a la función MQCRTMH. Llame a MQSETMP especificando este manejador de mensajes para cada propiedad que desee establecer. Se proporciona un programa de ejemplo, amqsstma.c, para ilustrar el uso de MQSETMP.

Si es un nuevo mensaje, cuando lo coloque en una cola utilizando MQPUT o MQPUT1, establezca el campo OriginalMsgHandle de MQPMO en el valor de este manejador de mensajes y establezca el campo Acción de MQPMO en MQACTP_NEW (este es el valor predeterminado).

Si es un mensaje que ha recuperado previamente y ahora está reenviándolo, respondiéndolo o enviando un informe como respuesta, coloque el manejador de mensajes original en el campo OriginalMsgHandle de MQPMO y el nuevo manejador de mensajes en el campo NewMsgHandle. Establezca el campo Acción en MQACTP_FORWARD, MQACTP_REPLY o MQACTP_REPORT, según corresponda.

Si tiene propiedades en una cabecera MQRFH2 de un mensaje que ha recuperado previamente, puede convertirlas en propiedades del manejador de mensajes utilizando la llamada MQBUFMH.

Si está colocando el mensaje en una cola de un gestor de colas de un nivel anterior a IBM WebSphere MQ 7.0 que no puede procesar las propiedades de mensaje, puede establecer el parámetro PropertyControl en la definición de canal para especificar cómo se deben tratar las propiedades.

Control de la información de contexto de mensaje

Cuando utilice la llamada MQPUT o MQPUT1 para colocar un mensaje en una cola, puede especificar que el gestor de colas añada alguna información de contexto predeterminada en el descriptor de mensaje. Las aplicaciones que tengan el nivel apropiado de autorización pueden añadir información de contexto adicional. Puede utilizar el campo de opciones de la estructura MQPMO para controlar la información de contexto.

La información de contexto de mensaje permite a la aplicación recuperar el mensaje para obtener información acerca de quién ha originado el mensaje. Toda la información de contexto se almacena en los campos de contexto del descriptor de mensaje. El tipo de información incluye la información de contexto de identidad, origen y usuario.

Para controlar la información de contexto, utilice el campo *Options* de la estructura MQPMO.

Si no especifica ninguna opción para la información de contexto, el gestor de colas sobrescribe la información de contexto que pueda existir en el descriptor de mensaje con la información de identidad y contexto generada para su mensaje. Esto es lo mismo que especificar la opción MQPMO_DEFAULT_CONTEXT.Es posible que desee esta información de contexto predeterminada al crear un nuevo mensaje (por ejemplo, cuando procese la entrada de usuario procedente de una pantalla de consulta).

Si no desea ninguna información de contexto asociada al mensaje, utilice la opción MQPMO_NO_CONTEXT. Al transferir un mensaje sin contexto, todas las comprobaciones de autorización que lleva a cabo IBM MQ se realizan utilizando un ID de usuario en blanco. No se puede asignar a un ID de usuario en blanco autorización explícita para los recursos de IBM MQ, pero se le trata como a un miembro del grupo especial 'nobody'. Para obtener más detalles sobre el grupo especial nobody, consulte Información de consulta sobre la interfaz de servicios instalables.

Puede establecer el contexto utilizando MQOPEN seguido por MQPUT usando la opción MQOO_ y la opción MQPMO_ indicadas en las secciones siguientes. También puede establecer el contexto utilizando simplemente MQPUT1, en cuyo caso necesitará seleccionar la opción MQPMO_ indicada en las secciones siguientes.

En las secciones siguientes de este tema se explica el uso del contexto de identidad, el contexto de usuario y todo el contexto.

- “Cómo pasar el contexto de identidad” en la página 780
- “Cómo pasar el contexto de usuario” en la página 780
- “Cómo pasar todo el contexto” en la página 780
- “Cómo establecer el contexto de identidad” en la página 780
- “Cómo establecer el contexto de usuario” en la página 781

- "Cómo establecer todo el contexto" en la página 781

Cómo pasar el contexto de identidad

En general, los programas deben pasar información de contexto de identidad de mensaje a mensaje, en lo relativo a una aplicación, hasta que los datos llegue a su destino final.

Los programas deben cambiar la información de contexto de origen cada vez que cambian los datos. No obstante, las aplicaciones que desean cambiar o establecer la información de contexto deben tener el nivel de autorización adecuado. El gestor de colas comprueba esta autorización cuando las aplicaciones abren las colas; deben tener autorización para poder utilizar las opciones de contexto adecuadas para la llamada MQOPEN.

Si la aplicación obtiene un mensaje, procesa los datos del mismo y, a continuación, coloca los datos cambiados en otro mensaje (posiblemente para que los procese otra aplicación), la aplicación debe pasar la información de contexto de identidad del mensaje original al nuevo mensaje. Puede permitir que el gestor de colas cree la información de contexto de origen.

Para guardar la información de contexto del mensaje original, utilice la opción MQOO_SAVE_ALL_CONTEXT cuando abra la cola para obtener el mensaje. Esto es adicional a las demás opciones que puede utilizar con la llamada MQOPEN. Tenga en cuenta, no obstante, que no puede guardar información de contexto si sólo examina el mensaje.

Cuando cree el segundo mensaje:

- Abra la cola utilizando la opción MQOO_PASS_IDENTITY_CONTEXT (además de la opción MQOO_OUTPUT).
- En el campo *Context* de la estructura de opciones de transferencia de mensaje, proporcione el manejador de la cola de la que se haya guardado la información de contexto.
- En el campo *Options* de la estructura de opciones de transferencia de mensaje, especifique la opción MQPMO_PASS_IDENTITY_CONTEXT.

Cómo pasar el contexto de usuario

No puede elegir pasar sólo el contexto de usuario. Para pasar el contexto de usuario al transferir un mensaje, especifique MQPMO_PASS_ALL_CONTEXT. Todas las propiedades del contexto de usuario se pasan del mismo modo que el contexto de origen.

Cuando tenga lugar una llamada MQPUT o MQPUT1 y se pasa el contexto, todas las propiedades del contexto de usuario se pasan del mensaje recuperado al mensaje transferido. Todas las propiedades de contexto de usuario que haya modificado la aplicación que efectúa la transferencia, se transfieren con sus valores originales. Todas las propiedades de contexto de usuario que haya suprimido la aplicación que efectúa la transferencia, se restauran en el mensaje transferido. Se conservan todas las propiedades de contexto de usuario que haya añadido al mensaje la aplicación que efectúa la transferencia.

Cómo pasar todo el contexto

Si la aplicación obtiene un mensaje, y transfiere los datos del mismo (sin cambiarlos) a otro mensaje, la aplicación debe pasar toda la información de contexto (identidad, origen y usuario) del mensaje original al nuevo mensaje. Un ejemplo de una aplicación que puede hacer esto es un transportador de mensajes, que mueve los mensajes de una cola a otra.

Siga el mismo procedimiento que para pasar el contexto de identidad, excepto que debe utilizar la opción MQOO_PASS_ALL_CONTEXT y la opción de transferencia de mensaje MQPMO_PASS_ALL_CONTEXT.

Cómo establecer el contexto de identidad

Si desea establecer la información del contexto de identidad de un mensaje:

- Abra la cola utilizando la opción MQOO_SET_IDENTITY_CONTEXT.

- Transfiera el mensaje a la cola, especificando la opción MQPMO_SET_IDENTITY_CONTEXT. En el descriptor de mensaje, especifique la información del contexto de identidad que necesite.

Nota: Cuando establezca algunos (pero no todos) los campos del contexto de identidad mediante las opciones MQOO_SET_IDENTITY_CONTEXT y MQPMO_SET_IDENTITY_CONTEXT, es importante que tenga en cuenta que el gestor de colas no establece ninguno de los demás campos.

Para poder modificar cualquier opción de contexto de mensaje, debe tener las autorizaciones apropiadas para emitir la llamada. Por ejemplo, para poder utilizar MQOO_SET_IDENTITY_CONTEXT o MQPMO_SET_IDENTITY_CONTEXT, debe tener el permiso `+setid`.

Cómo establecer el contexto de usuario

Para establecer una propiedad en el contexto de usuario, establezca el campo Context del descriptor de propiedad de mensaje (MQPD) en MQPD_USER_CONTEXT cuando efectúe la llamada MQSETMP.

No necesita ninguna autorización especial para poder establecer una propiedad en el contexto de usuario. El contexto de usuario no tiene las opciones MQOO_SET_* ni MQPMO_SET_*.

Cómo establecer todo el contexto

Si desea establecer la información del contexto de identidad y de origen de un mensaje:

1. Abra la cola utilizando la opción MQOO_SET_ALL_CONTEXT.
2. Transfiera el mensaje a la cola, especificando la opción MQPMO_SET_ALL_CONTEXT. En el descriptor de mensaje, especifique la información del contexto de identidad y de origen que necesite.

Se necesita la autorización adecuada para poder establecer cada tipo de valor de contexto.

Conceptos relacionados

"Contexto de mensaje" en la página 48
La información del *contexto de mensaje* permite a la aplicación que recupera el mensaje averiguar quién ha originado el mensaje.

Referencia relacionada

"Opciones de MQOPEN relacionadas con el contexto del mensaje" en la página 769
Si desea poder asociar información de contexto a un mensaje cuando lo coloca en una cola, debe utilizar una de las opciones de contexto de mensaje al abrir la cola.

Colocar un mensaje en una cola utilizando la llamada MQPUT1

Utilice la llamada MQPUT1 cuando desee cerrar la cola inmediatamente después de haber transferida a la misma un único mensaje. Por ejemplo, es probable que una aplicación de servidor utilice la llamada MQPUT1 cuando envía una respuesta a cada una de las distintas colas.

MQPUT1 es funcionalmente equivalente a la llamada MQOPEN seguida de MQPUT, seguida de MQCLOSE. La única diferencia en la sintaxis de las llamadas MQPUT y MQPUT1 es que en el caso de MQPUT especifica un descriptor de objeto, mientras que en MQPUT1 especifica una estructura de descriptor de objeto (MQOD), como se ha definido en MQOPEN. Consulte "Identificación de objetos (la estructura de MQOD)" en la página 763. Esto es debido a que es necesario proporcionar información a la llamada MQPUT1 sobre la cola que debe abrir, mientras que cuando se llama a MQPUT, la cola ya debe estar abierta.

Como entrada para la llamada MQPUT1, debe proporcionar lo siguiente:

- Un descriptor de conexión.
- Una descripción del objeto que desea abrir. Debe tener el formato de una estructura de descriptor de objeto (MQOD).
- Una descripción del mensaje que desea colocar en la cola. Tiene el formato de una estructura de descriptor de mensaje (MQMD).
- La información de control con el formato de una estructura de opciones de colocación de mensaje (MQPMO).

- La longitud de los datos contenidos dentro del mensaje (MQLONG).
- La dirección de los datos del mensaje.

La salida de MQPUT1 es:

- Un código de terminación
- Un código de razón

Si la llamada se completa de forma satisfactoria, también devuelve la estructura de las opciones y la estructura del descriptor de mensaje. La llamada modifica la estructura de las opciones para mostrar el nombre de la cola y el gestor de colas donde se envió el mensaje. Si solicita que el gestor de colas genere un valor exclusivo para el identificador del mensaje que está transfiriendo (especificando un cero binario en el campo *MsgId* de la estructura MQMD), la llamada inserta el valor en el campo *MsgId* antes de devolverle esta estructura.

Nota: No puede utilizar MQPUT1 con un nombre de cola modelo; no obstante, una vez que se ha abierto una cola, puede emitir una MQPUT1 a la cola dinámica.

Los seis parámetros de entrada para MQPUT1 son:

Hconn
Un descriptor de conexión. Para las aplicaciones CICS, puede especificar la constante MQHC_DEF_HCONN, cuyo valor es cero, o puede utilizar el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX. En el caso de otros programas, utilice el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX.

ObjDesc
Es una estructura de descriptor de objeto (MQOD).

En los campos *ObjectName* y *ObjectQMgrName*, especifique el nombre de la cola en la que desee colocar un mensaje, y el nombre del gestor de colas que posea dicha cola.

El campo *DynamicQName* se omite para la llamada MQPUT1, porque no puede utilizar colas modelo.

Utilice el campo *AlternateUserId* si desea nominar un identificador de usuario alternativo que deba utilizarse para probar si la autorización puede abrir la cola.

MsgDesc
Es una estructura de descriptor de mensaje (MQMD). Al igual que ocurre con la llamada MQPUT, utilice esta estructura para definir el mensaje que está transfiriendo a la cola.

PutMsgOpts
Es una estructura de opciones de transferencia de mensaje (MQPMO). Utilícelo como lo haría para una llamada MQPUT. Consulte “Especificación de opciones utilizando la estructura MQPMO” en la página 774.

Cuando se establece en cero el campo *Options*, el gestor de colas utiliza su propio ID de usuario cuando realiza pruebas de autorización de acceso a la cola. Además, el gestor de colas ignora cualquier identificador de usuario alternativo que se haya especificado en el campo *AlternateUserId* de la estructura MQOD.

BufferLength
Es la longitud del mensaje.

Buffer
Es el área de almacenamiento intermedio que contiene el texto del mensaje.

Si utiliza clústeres, MQPUT1 funciona como si MQOO_BIND_NOT_FIXED estuviera en vigor. Las aplicaciones deben utilizar los campos resueltos en la estructura MQPMO, en lugar de la estructura MQOD, para determinar dónde se envió el mensaje. Consulte Configuración de un clúster de gestor de colas para obtener más información.

En MQPUT1 se proporciona una descripción de la llamada MQPUT1.

Listas de distribución

No está soportado en IBM MQ for z/OS. Las listas de distribución permiten transferir un mensaje a varios destinos en una sola llamada MQPUT o MQPUT1.Una sola llamada MQOPEN puede abrir varias colas y, a continuación, una sola llamada MQPUT puede transferir un mensaje a cada una de dichas colas. Parte de la información genérica de las estructuras MQI utilizada para este proceso se puede reemplazar por información específica relativa a los destinos individuales incluidos en la lista de distribución.

Atención: Las listas de distribución no admiten el uso de cola alias que apuntan a objetos de tema. Si una cola de alias apunta a un objeto de tema en una lista de distribución, IBM MQ devuelve `MQRC_ALIAS_BASE_Q_TYPE_ERROR`.

Cuando se emite una llamada MQOPEN, se obtiene información genérica del descriptor de objetos (MQOD). Si especifica MQOD_VERSION_2 en el campo *`Version`* y especifica un valor mayor que cero en el campo *`RecsPresent`*, el valor de *`Hobj`* se puede definir como un manejador de una lista (de una o más colas) en lugar de una cola. En este caso, la información específica se proporcionan a través de los registros de objeto (MQOR), que proporcionan detalles del destino (es decir, *`ObjectName`* y *`ObjectQMgrName`*).

El manejador de objeto (*`Hobj`*) se pasa a la llamada MQPUT, lo que permite colocarlo en una lista, en lugar de en una sola cola.

Cuando se transfiere un mensaje a las colas (MQPUT), se obtiene información genérica de la estructura de la opción de transferencia de mensaje (MQPMO) y del descriptor de mensaje (MQMD). Se proporciona información específica en forma de registros de transferencia de mensaje (MQPMR).

Los registros de respuesta (MQRR) pueden recibir un código de terminación y un código de razón específicos para cada cola de destino.

En la Figura 56 en la página 783, se muestra cómo funcionan las listas de distribución.

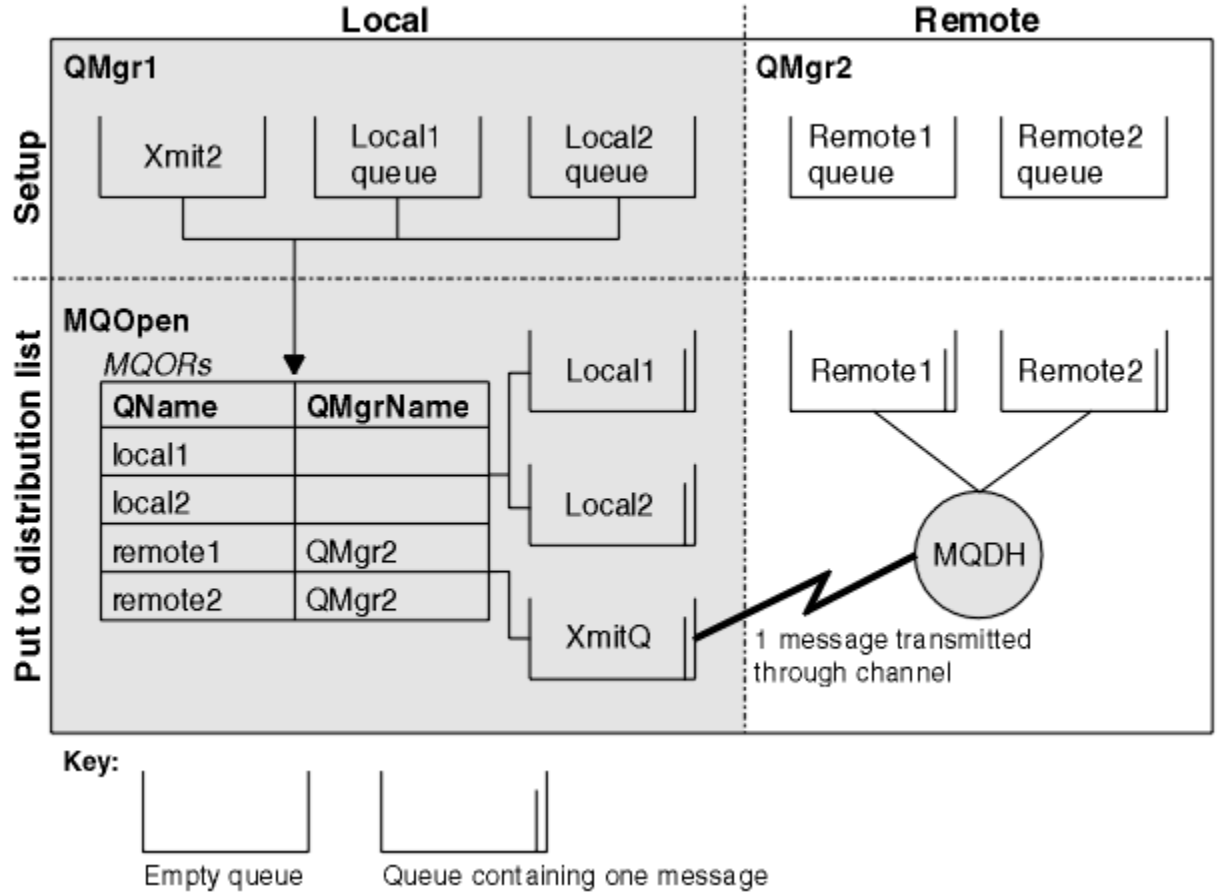


Figura 56. Cómo funcionan las listas de distribución

Apertura de una lista de distribución
Utilice la llamada MQOPEN para abrir una lista de distribución y las opciones de la llamada para especificar lo que desea hacer con la lista.

Como entrada de MQOPEN, hay que proporcionar:

- Un manejador de conexión (consulte "Colocación de mensajes en una cola" en la página 772 para obtener una descripción).
- Información genérica en la estructura de Descriptor de Objeto (MQOD).
- El nombre de las colas que desee abrir, utilizando la estructura de Registro de Objetos (MQOR).

La salida de MQOPEN es:

- Un manejador de objetos que representa el acceso a la lista de distribución.
- Un código de terminación genérico.
- Un código de razón genérico.
- Registros de respuesta (opcionales) que contienen un código de terminación y una razón por cada destino.

Uso de la estructura MQOD

Utilice la estructura MQOD para identificar las colas que desee abrir.

Para definir una lista de distribución, hay que especificar MQOD_VERSION_2 en el campo *Version*, un valor mayor que cero en el campo *RecsPresent* y MQOT_Q en el campo *ObjectType*. Consulte MQOD para obtener una descripción de todos los campos de la estructura MQOD.

Uso de la estructura MQOR

Proporcione una estructura MQOR por cada destino.

La estructura contiene los nombres del gestor de colas y de la cola de destino. Los campos *ObjectName* y *ObjectQMgrName* en la MQOD no se usan en listas de distribución. Tiene que haber uno o más registros de objeto. Si *ObjectQMgrName* se deja en blanco, se utiliza el gestor de colas local. Consulte ObjectName y ObjectQMgrName para obtener información estos campos.

Las colas de destino se pueden especificar de dos maneras:

- Utilizando el campo de desplazamiento *ObjectRecOffset*.

  En este caso, la aplicación debe declarar su propia estructura que contenga una estructura MQOD, seguida de la matriz de registros MQOR (con tantos elementos de matriz como sean necesarios) y establecer *ObjectRecOffset* en el desplazamiento del primer elemento de la matriz desde el inicio de MQOD. Asegúrese de que este desplazamiento sea correcto.

  Se recomienda el uso de los recursos proporcionados por el lenguaje de programación, si estos están disponibles en todos los entornos en los que ejecuta la aplicación El código siguiente ilustra esta técnica en 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.
```

  De forma alternativa, utilice la constante MQOD_CURRENT_LENGTH si el lenguaje de programación no soporta los recursos incorporados necesarios en todos los entornos implicados. El código siguiente ilustra esta técnica:

```
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.
```

Sin embargo, esto sólo funciona correctamente si la estructura MQOD y la matriz de registros MQOR son contiguos; si el compilador inserta bytes de omisión entre MQOD y la matriz MQOR, estos se deben añadir al valor almacenado en *ObjectRecOffset*.

Se recomienda utilizar *ObjectRecOffset* para lenguajes de programación que no dan soporte al tipo de datos de puntero, o que implementan el tipo de datos de puntero de una forma que no es portable a entornos diferentes (por ejemplo, el lenguaje de programación COBOL).

- Utilizando el campo de puntero *ObjectRecPtr*.

  En este caso, la aplicación puede declarar la matriz de estructuras MQOR por separado de la estructura MQOD y establecer *ObjectRecPtr* en la dirección de la matriz. El código siguiente ilustra esta técnica en C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  Se recomienda utilizar *ObjectRecPtr* para los lenguajes de programación que dan soporte al tipo de datos de puntero de una forma que es portable a distintos entornos (por ejemplo, el lenguaje de programación C).

Sea cual sea la técnica escogida, hay que usar o *ObjectRecOffset* o *ObjectRecPtr*; la llamada fallará con código de razón MQRC_OBJECT_RECORDS_ERROR si ambos son cero o distintos de cero.

Uso de la estructura MQRR

Estas estructuras son específicas del destino; cada registro de respuesta contiene un campo *CompCode* y *Reason* para cada cola de una lista de distribución. Hay que usar esta estructura para poder determinar dónde se encuentran los problemas.

Por ejemplo, si se recibe un código de razón de MQRC_MULTIPLE_REASONS y la lista de distribución contiene cinco colas de destino, no podrá saberse a qué colas se refieren los problemas si no se utiliza esta estructura. Sin embargo, si tiene un código de terminación y un código de razón por cada destino, se podrán localizar los errores con más facilidad.

Consulte MQRR para obtener más información sobre la estructura MQRR.

En Figura 57 en la página 785 se muestra cómo se puede abrir una lista de distribución en C.

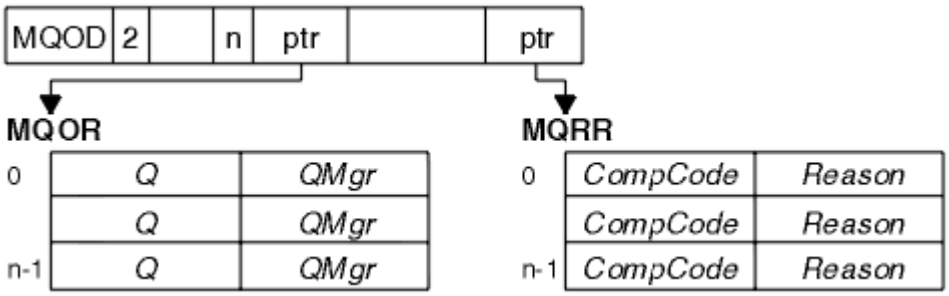


Figura 57. Apertura de una lista de distribución en C

En Figura 58 en la página 786 se muestra cómo se puede abrir una lista de distribución en COBOL.

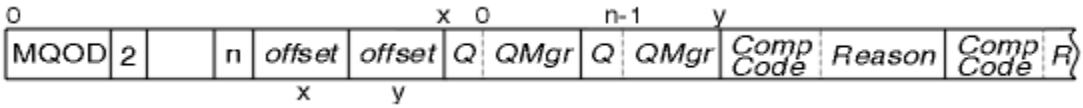


Figura 58. Apertura de una lista de distribución en COBOL

Uso de las opciones de MQOPEN

Cuando se abre una lista de distribución, se pueden especificar las opciones siguientes:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (opcional)
- MQOO_ALTERNATE_USER_AUTHORITY (opcional)
- MQOO_*_CONTEXT (opcional)

Consulte “Apertura y cierre de objetos” en la página 761 para obtener una descripción de estas opciones.

Colocación de mensajes en una lista de distribución
Para colocar mensajes en una lista de distribución, se puede utilizar MQPUT o MQPUT1.

Como entrada, hay que facilitar:

- Un descriptor de conexión (consulte “Colocación de mensajes en una cola” en la página 772 para obtener una descripción).
- Un descriptor de objeto. Si se abre una lista de distribución utilizando MQOPEN, el el *Hobj* solo permite colocar en la lista.
- Una estructura de descriptor de mensaje (MQMD). Consulte MQMD para ver una descripción de esta estructura.
- La información de control en formato de una estructura de opciones de colocación de mensaje (MQPMO). Consulte “Especificación de opciones utilizando la estructura MQPMO” en la página 774 para obtener información sobre cómo completar los campos de la estructura MQPMO.
- Información de control en forma de registros de colocación de mensajes (Put Message Records, MQPMR).
- La longitud de los datos contenidos dentro del mensaje (MQLONG).
- Los propios datos del mensaje.

La salida es:

- Un código de terminación
- Un código de razón
- Registros de respuesta (opcional).

Uso de la estructura MQPMR

Esta estructura es opcional y proporciona información específica de destino para algunos campos que puede que se quieran identificar de forma distinta a los que ya están identificados en el MQMD.

Para obtener una descripción de estos campos, consulte MQPMR.

El contenido de cada registro depende de la información proporcionada en el campo *PutMsgRecFields* de MQPMO. Por ejemplo, en el programa de ejemplo AMQSPTL0.C (consulte “El programa de ejemplo de lista de distribución” en la página 1117 para obtener una descripción) que ilustra el uso de las listas de distribución, el ejemplo opta por proporcionar valores para *MsgId* y *CorrelId* en el MQPMR. Esta sección del programa de ejemplo es similar a la siguiente:

```
typedef struct
{
MQBYTE24 MsgId;
```

```
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

Esto implica que se proporcionan *MsgId* y *CorrelId* para cada destino de una lista de distribución. Los registros de mensajes de colocación se proporcionan como un vector.

Figura 59 en la página 787 muestra cómo se puede colocar un mensaje en una lista de distribución en C.

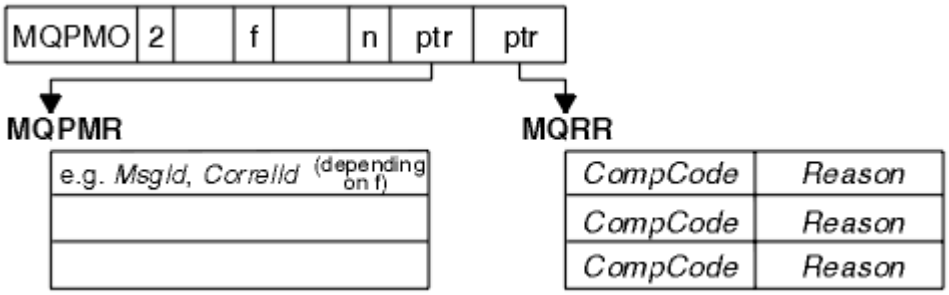


Figura 59. Colocación de un mensaje en una lista de distribución en C

Figura 60 en la página 787 muestra cómo se puede colocar un mensaje en una lista de distribución en COBOL.

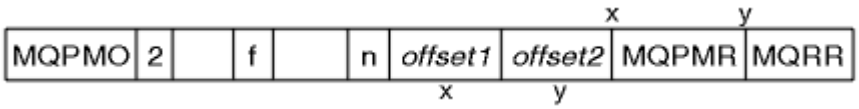


Figura 60. Colocación de un mensaje en una lista de distribución en COBOL

Utilización de MQPUT1

Si está utilizando MQPUT1, tenga en cuenta los puntos siguientes:

1. Los valores de los campos *ResponseRecOffset* y *ResponseRecPtr* deben ser nulos o cero.
2. Los registros de respuesta, en caso de ser necesarios, tienen que referenciarse en la MQOD.

Algunos casos en los que las llamadas put fallan

Si se cambian determinados atributos de una cola usando la opción FORCE en un mandato durante el intervalo de la emisión de un MQOPEN y una llamada MQPUT, la llamada MQPUT falla, y devuelve el código de razón MQRC_OBJECT_CHANGED.

El gestor de colas marca el descriptor de objeto como no válido. Esto también ocurre si los cambios se realizan mientras se procesa una llamada MQPUT1, o si los cambios se aplican a cualquier cola a la que se resuelve el nombre de la cola. Los atributos que afectan al descriptor de este modo se listan en la descripción de la llamada MQOPEN contenida en la sección MQOPEN. Si la llamada devuelve el código de razón MQRC_OBJECT_CHANGED, cierre la cola, vuelva a abrirla y vuelva a intentar colocar un mensaje.

Si las operaciones de colocación (put) se inhiben para una cola en la que está intentando colocar mensajes (o cualquier cola en la que se resuelve el nombre de la cola), la llamada MQPUT o MQPUT1 falla y devuelve el código de razón MQRC_PUT_INHIBITED. Es posible que pueda poner un mensaje correctamente si intenta realizar la llamada en un momento posterior, si el diseño de la aplicación es tal que otros programas cambian los atributos de las colas con frecuencia.

Además, si la cola en la que está intentando poner el mensaje está llena, la llamada MQPUT o MQPUT1 falla y devuelve MQRC_Q_FULL.

Si se ha suprimido una cola dinámica (temporal o permanente), las llamadas MQPUT que utilizan un manejador de objeto adquirido previamente fallan y devuelven el código de razón MQRC_Q_DELETED. En esta situación, se recomienda cerrar el descriptor de objeto ya que no es útil.

En el caso de las listas de distribución, se pueden producir varios códigos de terminación y códigos de razón en una única solicitud. Esto no se puede manejar utilizando únicamente los campos de salida *CompCode* y *Reason* en MQOPEN y MQPUT.

Cuando utiliza listas de distribución para colocar mensajes en varios destinos, los registros de respuesta contienen el *CompCode* y *Reason* específicos para cada destino. Si recibe un código de terminación de MQCC_FAILED, no se coloca correctamente ningún mensaje en ninguna cola de destino. Si el código de terminación es MQCC_WARNING, el mensaje se coloca correctamente en una o más de las colas de destino. Si recibe un código de retorno de MQRC_MULTIPLE_REASONS, los códigos de razón no son todos los mismos para todos los destinos. Por lo tanto, se recomienda utilizar la estructura MQRR de modo que pueda determinar qué cola o colas han causado un error y las razones de cada uno.

Obtención de mensajes de una cola

Utilice esta información para aprender a obtener mensajes de una cola.

Puede obtener mensajes de una cola de dos maneras:

1. Puede eliminar un mensaje de la cola para que otros programas ya no puedan verlo.
2. Puede copiar un mensaje, dejando el mensaje original en la cola. Esto se conoce como *examinar*. Puede eliminar el mensaje una vez se haya examinado.

En ambos casos, utilice la llamada MQGET, pero primero la aplicación debe estar conectada al gestor de colas, y debe utilizar la llamada MQOPEN para abrir la cola (para especificarlos, examinarlos o ambos). Estas operaciones se describen en “Conexión y desconexión de un gestor de colas” en la página 754 y “Apertura y cierre de objetos” en la página 761.

Cuando haya abierto la cola, puede utilizar repetidamente la llamada MQGET para examinar o eliminar mensajes en la misma cola. Llame a MQCLOSE cuando haya terminado de obtener todos los mensajes que desee de la cola.

Utilice los siguientes enlaces para obtener más información sobre cómo obtener mensajes de una cola:

- “Obtener mensajes de una cola utilizando la llamada MQGET” en la página 789
- “Orden en el que se recuperan los mensajes de una cola” en la página 793
- “Obtener un mensaje concreto” en la página 806
- “Mejora del rendimiento de mensajes no persistentes” en la página 807
- z/OS “Tipo de índice” en la página 811
- “Manejo de mensajes de más de 4 MB de tamaño” en la página 812
- “Espera de mensajes” en la página 818
- z/OS “Uso de señales” en la página 818
- “Omisión de restitución” en la página 820
- “Conversión de datos de aplicación” en la página 822
- “Cómo examinar mensajes en una cola” en la página 823
- “Algunos casos en los que falla la llamada MQGET” en la página 829

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772

Utilice esta información para conocer cómo poner mensajes en una cola.

"Consulta y establecimiento de atributos de objeto" en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Obtener mensajes de una cola utilizando la llamada MQGET

La llamada MQGET obtiene un mensaje de una cola local abierta. No puede obtener un mensaje de una cola de otro sistema.

Como entrada a la llamada MQGET, debe proporcionar:

- Un descriptor de conexión.
- Un manejador de cola.
- Una descripción del mensaje que desea obtener de la cola. Tiene el formato de una estructura de descriptor de mensaje (MQMD).
- Información de control con el formato de una estructura de opciones de transferencia de mensaje (MQGMO).
- El tamaño del almacenamiento intermedio que ha asignado para el mensaje (MQLONG).
- La dirección del almacén en el que se ha de colocar el mensaje.

La salida de MQGET es:

- Un código de razón
- Un código de terminación
- El mensaje del área de almacenamiento intermedio que ha especificado, si la llamada se completa correctamente.
- Su estructura de opciones, modificada para que se muestre el nombre de la cola desde la que se ha recuperado el mensaje
- Su estructura del descriptor de mensaje, con el contenido de los campos modificados para describir el mensaje que se ha recuperado
- La longitud del mensaje (MQLONG)

Es una descripción de la llamada MQGET en MQGET.

Las secciones siguientes describen la información que debe proporcionar como entrada para la llamada MQGET.

- "Especificar descriptores de conexión" en la página 790
- "Descripción de mensajes utilizando la estructura MQMD y la llamada MQGET" en la página 790
- "Especificar opciones MQGET utilizando la estructura MQGMO" en la página 790

- “Especificar el tamaño del área de almacenamiento intermedio” en la página 792

Especificar descriptores de conexión

z/OS En el caso de las aplicaciones CICS en z/OS, puede especificar la constante MQHC_DEF_HCONN (cuyo valor es cero) o utilizar el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX. Para otras aplicaciones, utilice siempre el descriptor de conexión que devuelve la llamada MQCONN o MQCONNX.

Utilice el descriptor de conexión (*Hobj*) que se devuelve cuando invoca MQOPEN.

Descripción de mensajes utilizando la estructura MQMD y la llamada MQGET

Para identificar el mensaje que desea obtener de una cola, utilice la estructura del descriptor de mensajes (MQMD).

Se trata de un parámetro de entrada/salida para la llamada MQGET. Hay una introducción a las propiedades de mensaje que MQMD describe en “Mensajes de IBM MQ” en la página 18, y existe una descripción de la propia estructura en MQMD.

Si sabe qué mensaje de la cola desea obtener, consulte “Obtener un mensaje concreto” en la página 806.

Si no especifica un mensaje concreto, MQGET recupera el *primer* mensaje de la cola. En la sección “Orden en el que se recuperan los mensajes de una cola” en la página 793 se describe cómo la prioridad de un mensaje, el atributo **MsgDeliverySequence** de una cola y la opción MQGMO_LOGICAL_ORDER determina el orden de los mensajes en la cola.

Nota: Si desea utilizar MQGET más de una vez, por ejemplo, para recorrer cada uno de los mensajes de la cola, debe establecer los campos *MsgId* y *CorrelId* de esta estructura en valores nulos después de cada llamada. De este modo, se borran estos campos de los identificadores del mensaje que se ha recuperado.

No obstante, si desea agrupar sus mensajes, el valor de *GroupId* debe ser el mismo para los mensajes del mismo grupo, de modo que la llamada busque un mensaje con el mismo identificador que el mensaje anterior y así cubrir el grupo completo.

Especificar opciones MQGET utilizando la estructura MQGMO

La estructura MQGMO es una variable de entrada/salida para pasar opciones a la llamada MQGET. Las siguientes secciones le ayudan a completar algunos de los campos de esta estructura.

Puede encontrar una descripción de la estructura MQGMO en la sección MQGMO.

StrucId
StrucId es un campo de 4 caracteres que se utiliza para identificar la estructura como una estructura de opciones de obtención de mensajes. Especifique siempre MQGMO_STRUC_ID.

Version
*Version*describe el número de versión de la estructura. El valor predeterminado es MQGMO_VERSION_1. Si desea utilizar los campos de la Versión 2 o recuperar mensajes por orden lógico, especifique MQGMO_VERSION_2. Si desea utilizar los campos de la Versión 3 o recuperar mensajes por orden lógico, especifique MQGMO_VERSION_3. MQGMO_CURRENT_VERSION establece su aplicación para que utilice el nivel más reciente.

Options
Dentro del código, puede seleccionar las opciones en cualquier orden; cada opción se representa mediante un bit en el campo *Options* .

El campo *Options* controla:

- Si la llamada MQGET espera a que llegue un mensaje a la cola antes de completarse. Consulte “Espera de mensajes” en la página 818

- Si se incluye la operación de obtención en una unidad de trabajo.
- Si se recupera un mensaje no persistente fuera del punto de sincronización, lo que permite la mensajería rápida
- z/OS En IBM MQ for z/OS, si se recupera el mensaje que se ha marcado para omitir restitución. Consulte “Omisión de restitución” en la página 820
- Si se ha eliminado el mensaje de la cola o simplemente se ha explorado
- Si se ha de seleccionar un mensaje utilizando un cursor para examinar o mediante otro criterio de selección
- Si la llamada se realiza correctamente, incluso si el mensaje ya no está en su almacenamiento intermedio
- z/OS En IBM MQ for z/OS, si se permite que se complete la llamada. Esta opción también establece una señal que indica que desea que se le notifique la llegada de un mensaje.
- Si la llamada falla si el gestor de colas está en estado de desactivación temporal
- z/OS En IBM MQ for z/OS, si falla la llamada si la conexión está en estado de desactivación temporal
- Si es necesaria la conversión de datos del mensaje de aplicación. Consulte “Conversión de datos de aplicación” en la página 822
- El orden en que se recuperan los mensajes y segmentos de una cola de z/OS (excepto para IBM MQ for z/OS)
- Si se completa, solo se pueden recuperar los mensajes lógicos z/OS (excepto para IBM MQ for z/OS)
- Si se pueden recuperar los mensajes de un grupo únicamente cuando están disponibles *todos* los mensajes del grupo
- Si se pueden recuperar los segmentos de un mensaje lógico únicamente cuando están disponibles *todos* los mensajes lógicos z/OS (excepto para IBM MQ for z/OS)

Si deja el campo *Options* establecido en el valor predeterminado (MQGMO_NO_WAIT), la llamada MQGET funciona de este modo:

- Si no hay ninguna coincidencia de mensajes para su criterio de selección en la cola, la llamada no espera a que llegue un mensaje pero se completa de forma inmediata. z/OS Asimismo, en IBM MQ for z/OS, la llamada no establece una señal para solicitar la notificación de la llegada de un mensaje de este tipo.
- La plataforma determina el modo en que funciona la llamada con los puntos de sincronización:

Plataforma	**Bajo control de punto de sincronización**
IBM i	No
Sistemas AIX and Linux	No
z/OS z/OS z/OS	Sí
Sistemas Windows	No

- z/OS En IBM MQ for z/OS, el mensaje recuperado no está marcado para omitir restitución.
- El mensaje seleccionado se elimina de la cola (no explorado).
- No se requiere ningún tipo de conversión de datos de mensaje de aplicación.
- La llamada falla si el mensaje ya no está en el almacenamiento intermedio.

WaitInterval
El campo *WaitInterval* especifica el periodo máximo de tiempo (en milisegundos) que espera la llamada MQGET a que llegue un mensaje a la cola, cuando utiliza la opción MQGMO_WAIT. Si no se llega ningún mensaje durante el periodo especificado en *WaitInterval*, se finaliza la llamada y se devuelve un código de razón que indica que ningún mensaje de la cola coincide con el criterio de selección.

z/OS En IBM MQ for z/OS, si utiliza la opción MQGMO_SET_SIGNAL, el campo *WaitInterval* especifica la hora a la que se establece la señal.

Para obtener más información sobre estas opciones, consulte los apartados "Espera de mensajes" en la página 818 z/OS y "Uso de señales" en la página 818 .

z/OS ***Signal1***
Signal1 sólo está soportado en IBM MQ for z/OS.

Si utiliza la opción MQGMO_SET_SIGNAL para solicitar que se le notifique a su aplicación la llegada de mensaje adecuado, especifique el tipo de señal en el campo *Signal1*. En IBM MQ en todas las demás plataformas, el campo *Signal1* está reservado y su valor no es significativo.

z/OS Para obtener más información, consulte "Uso de señales" en la página 818.

Signal2
El campo *Signal2* está reservado en todas las plataformas y su valor no es importante.

z/OS Para obtener más información, consulte "Uso de señales" en la página 818.

ResolvedQName
ResolvedQName es un campo de salida en el que el gestor de colas devuelve el nombre de la cola de la que se ha recuperado el mensaje, después de la resolución de cualquier alias.

MatchOptions
MatchOptions controla los criterios de selección utilizados para MQGET.

GroupStatus
GroupStatus indica si el mensaje que ha recuperado está en un grupo.

SegmentStatus
SegmentStatus indica si el elemento que ha recuperado es un segmento de un mensaje lógico.

Segmentation
Segmentation indica si se permite la segmentación en el mensaje recuperado.

MsgToken

MsgToken identifica un mensaje de forma exclusiva.

ReturnedLength
ReturnedLength es un campo de salida en el que el gestor de colas devuelve la longitud de los datos del mensaje devuelto, en bytes.

MsgHandle
El descriptor de un mensaje que se debe llenar con las propiedades del mensaje que se va a recuperar de la cola. El descriptor se ha creado previamente mediante una llamada MQCRTMH. Las propiedades que ya están asociadas al descriptor se borran antes de recuperar un mensaje.

Especificar el tamaño del área de almacenamiento intermedio

En el parámetro **BufferLength** de la llamada MQGET, especifique el tamaño del área de almacenamiento intermedio que contendrá los datos del mensaje que va a recuperar. Puede decidir el tamaño de tres formas:

1. Es posible que ya conozca la longitud de los mensajes que espera de este programa. Si es así, especifique un almacenamiento intermedio de este tamaño.

No obstante, puede utilizar la opción MQGMO_ACCEPT_TRUNCATED_MSG en la estructura MQGMO, si desea que la llamada MQGET se completo se complete, a pesar de que el mensaje sea demasiado grande para el almacenamiento intermedio. En este caso:

- El almacenamiento intermedio se rellena con la cantidad del mensaje que puede contener
- La llamada devuelve un código de terminación de aviso
- Se elimina el mensaje la cola, se descarta el resto del mensaje o, en el caso de que esté explorando la cola, el cursor para examinar avanza.
- La longitud real del mensaje se devuelve en *DataLength*

Sin esta opción, la llamada se continúa completando con un aviso pero no se elimina el mensaje de la cola ni se avanza el cursor para examinar.

2. Calcule un tamaño para el almacenamiento intermedio, o incluso especifique un tamaño de cero bytes, y *no* utilice la opción MQGMO_ACCEPT_TRUNCATED_MSG. Si falla la llamada MQGET, por ejemplo, debido a que el almacenamiento intermedio es demasiado pequeño, se devuelve la longitud del mensaje en el parámetro **DataLength** de la llamada. El almacenamiento intermedio se rellena con la cantidad del mensaje que puede contener, pero el proceso de la llamada no se completa. Guarde el *MsgId* de este mensaje, a continuación, repita la llamada MQGET especificando un área de almacenamiento intermedio del tamaño correcto y el *MsgId* que de la primera llamada que ha anotado.

   Si su programa da servicio a una cola a la que otros programas también prestan servicio, es posible que uno de estos otros programas elimine el mensaje que desea antes de que su mensaje pueda emitir otra llamada MQGET. Su programa puede perder tiempo buscando un mensaje que ya no existe. Para evitar esto, en primer lugar, explore la cola hasta que encuentre el mensaje que desea, especificando un valor de cero para *BufferLength* y utilizando la opción MQGMO_ACCEPT_TRUNCATED_MSG. Eso coloca el cursor para examinar debajo del mensaje que desea. A continuación, puede recuperar el mensaje con otra llamada MQGET que especifique la opción MQGMO_MSG_UNDER_CURSOR. Si otro programa elimina el mensaje durante sus llamadas de exploración y supresión, inmediatamente la segunda llamada MQGET sin buscar en toda la cola, ya que no hay ningún mensaje bajo su cursor para examinar.
3. El atributo de *cola MaxMsgLength* determina la longitud máxima de los mensajes aceptada para dicha cola. El atributo de *gestor de colas MaxMsgLength* determina la longitud máxima de los mensajes aceptada para dicho gestor de colas. Si desconoce la longitud del mensaje, puede realizar una consulte acerca del atributo **MaxMsgLength** utilizando la llamada MQINQ y, a continuación, puede especificar un almacenamiento de este tamaño.

   >Para evitar que disminuya el rendimiento, intente que el tamaño del almacenamiento intermedio sea prácticamente igual al tamaño del mensaje real.

   Para obtener más información acerca del atributo **MaxMsgLength**, consulte “Aumentar la longitud máxima del mensaje” en la página 812.

Orden en el que se recuperan los mensajes de una cola

Puede controlar el orden en el que recupera mensajes de una cola. Esta sección describe las opciones disponibles.

Prioridad

Un programa puede asignar una prioridad a un mensaje cuando coloca el mensaje en una cola (consulte “Prioridades de mensajes” en la página 27). Los mensajes de igual prioridad se almacenan en una cola en orden de llegada, no en el orden en el que se han confirmado.

El gestor de colas mantiene las colas en orden FIFO (primero en entrar, primero en salir) o en FIFO dentro de la secuencia de prioridad. Esto depende del valor del atributo **MsgDeliverySequence** de la cola. Cuando un mensaje llega a una cola, se inserta inmediatamente después del último mensaje que tenga la misma prioridad.

Los programas pueden obtener el primer mensaje de una cola, o pueden obtener un mensaje determinado de una cola, sin tener en cuenta la prioridad de esos mensajes. Por ejemplo, un programa puede desear procesar la respuesta a un mensaje determinado que el programa ha enviado antes. Para obtener más información, consulte “Obtener un mensaje concreto” en la página 806.

Si una aplicación pone una secuencia de mensajes en una cola, otra aplicación puede recuperar esos mensajes en el mismo orden en el que se colocaron, siempre que se cumplan estas condiciones:

- Todos los mensajes tienen la misma prioridad
- Los mensajes se colocaron todos dentro de la misma unidad de trabajo o se colocaron todos fuera de una unidad de trabajo
- La cola es local respecto a la aplicación que realiza la operación de colocación

Si estas condiciones no se cumplen, y las aplicaciones dependen de que los mensajes se recuperen en un orden determinado, las aplicaciones deben incluir información de secuenciación en los datos del mensaje o establecer un medio de reconocer la recepción de un mensaje antes de enviar el siguiente.

z/OS En IBM MQ for z/OS, puede utilizar el atributo de cola, *IndexType*, para aumentar la velocidad de las operaciones MQGET en la cola. Para obtener más información, consulte “Tipo de índice” en la página 811.

Ordenación lógica y física
Los mensajes de una cola pueden estar (dentro de cada nivel de prioridad) en orden *físico* o *lógico*.

El orden físico es el orden en el que los mensajes llegan a una cola. El orden lógico es cuando todos los mensajes y segmentos dentro de un grupo están en su secuencia lógica, unos a continuación de otros, en la posición determinada por la posición física del primer elemento perteneciente al grupo.

Para obtener una descripción de grupos, mensajes y segmentos, consulte “Grupos de mensajes” en la página 45. Estos órdenes físicos y lógicos pueden diferir debido a que:

- Los grupos pueden llegar a un destino en momentos similares procedentes de aplicaciones diferentes, por lo tanto, pierden cualquier orden físico diferenciado.
- Incluso dentro de un mismo grupo, los mensajes pueden perder el orden debido a la redirección o el retraso de algunos de los mensajes del grupo.

Por ejemplo, el orden lógico podría ser el mostrado en la figura Figura 61 en la página 795:

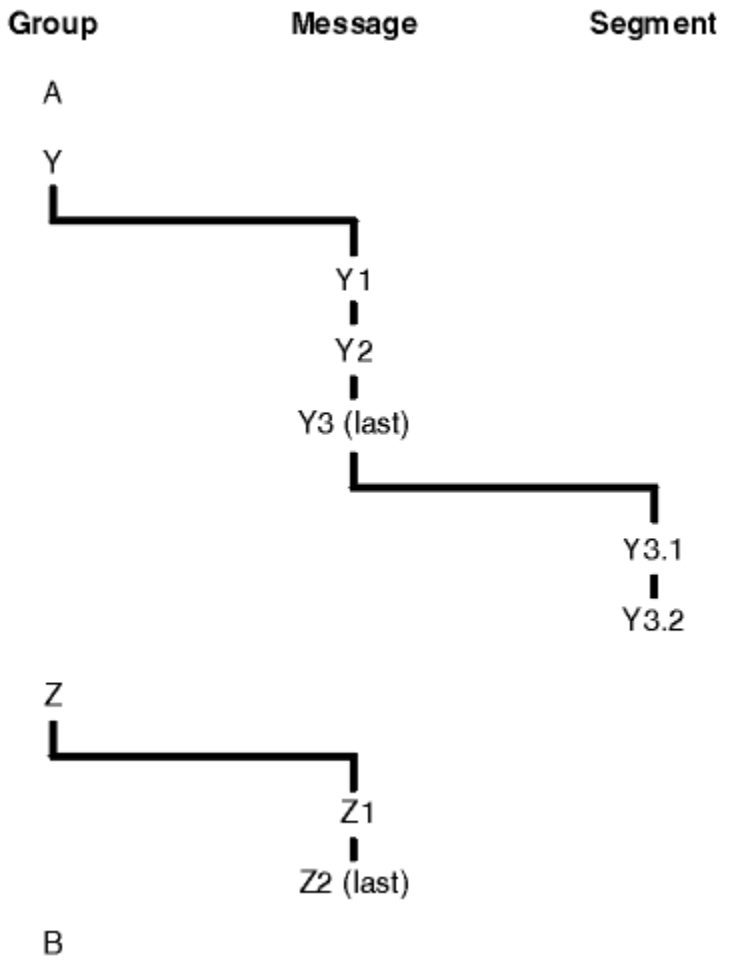


Figura 61. Orden lógico en una cola

Estos mensajes estarían en el orden lógico siguiente en una cola:

1. Mensaje A (no en un grupo)
2. Mensaje lógico 1 del grupo Y
3. Mensaje lógico 2 del grupo Y
4. Segmento 1 de (último) mensaje lógico 3 del grupo Y
5. (Último) segmento 2 del (último) mensaje lógico 3 del grupo Y
6. Mensaje lógico 1 del grupo Z
7. (Último) mensaje lógico 2 del grupo Z
8. Mensaje B (no en un grupo)

En cambio, el orden físico puede ser completamente diferente. La posición física del *primer* elemento dentro de cada grupo determina la posición lógica de todo el grupo. Por ejemplo, si los grupos Y y Z llegaron en momentos similares, y el mensaje 2 del grupo Z se puso delante del mensaje 1 del mismo grupo, el orden físico sería el mostrado en la figura Figura 62 en la página 796:

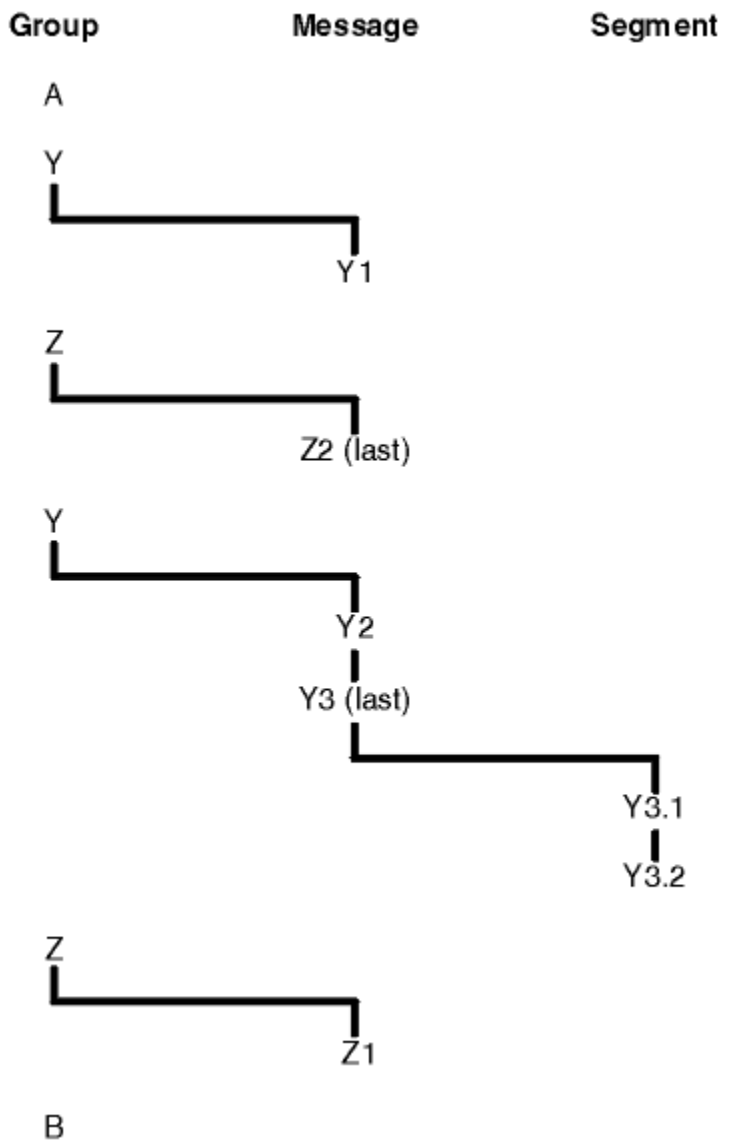


Figura 62. Orden físico en una cola

Estos mensajes estarían en el orden físico siguiente en la cola:

1. Mensaje A (no en un grupo)
2. Mensaje lógico 1 del grupo Y
3. Mensaje lógico 2 del grupo Z
4. Mensaje lógico 2 del grupo Y
5. Segmento 1 de (último) mensaje lógico 3 del grupo Y
6. (Último) segmento 2 del (último) mensaje lógico 3 del grupo Y
7. Mensaje lógico 1 del grupo Z
8. Mensaje B (no en un grupo)

Nota: En IBM MQ for z/OS, el orden físico de los mensajes de la cola no está garantizado si la cola está indexada por GROUPID.

Al obtener mensajes, puede especificar MQGMO_LOGICAL_ORDER para recuperar mensajes en orden lógico en lugar del orden físico.

Si emite una llamada MQGET con MQGMO_BROWSE_FIRST y MQGMO_LOGICAL_ORDER, las llamadas MQGET subsiguientes con MQGMO_BROWSE_NEXT deben también especificar

MQGMO_LOGICAL_ORDER. Por el contrario, si la llamada MQGET con MQGMO_BROWSE_FIRST no especifica MQGMO_LOGICAL_ORDER, tampoco deben especificarlo las llamadas MQGET subsiguientes con MQGMO_BROWSE_NEXT.

La información de grupo y segmento que el gestor de colas retiene para las llamadas MQGET que examinan mensajes de la cola está separada de la información de grupo y segmento que el gestor de colas retiene para las llamadas MQGET que eliminan mensajes de la cola. Cuando especifica MQGMO_BROWSE_FIRST, el gestor de colas pasa por alto la información de grupo y de segmento para el examen y explora la cola como si no hubiera ningún grupo actual y ningún mensaje lógico actual.

Nota: No utilice una llamada MQGET para examinar *más allá del final* de un grupo de mensajes (o mensaje lógico que no está en un grupo) sin especificar MQGMO_LOGICAL_ORDER. Por ejemplo, si el último mensaje del grupo *precede* al primer mensaje en el grupo de la cola, el uso de MQGMO_BROWSE_NEXT para examinar más allá del final del grupo, junto con la especificación MQMO_MATCH_MSG_SEQ_NUMBER y *MsgSeqNumber* establecido en 1 (para encontrar el primer mensaje del grupo siguiente) devuelve de nuevo el primer mensaje del grupo ya examinado. Esto podría ocurrir de forma inmediata, o después de varias llamadas MQGET (si hay grupos intermedios).

Evite la posibilidad de un bucle infinito abriendo la cola *dos veces* para examen:

- Utilice el primer descriptor para examinar solamente el primer mensaje de cada grupo.
- Utilice el segundo descriptor para examinar solamente los mensajes de un grupo específico.
- Utilice las opciones MQMO_* para desplazar el segundo cursor de examen a la posición del primer cursor de examen, antes de examinar los mensajes del grupo.
- No utilice el examen de MQGMO_BROWSE_NEXT más allá del final de un grupo.

Para obtener más información sobre este tema, consulte MQGET, MQMD y Reglas para validad opciones de MQI.

Para la mayoría de aplicaciones, probablemente eligirá el orden lógico o físico al examinar. Pero si desea conmutar entre estas modalidades, recuerde que cuando emite por primera vez un examen con MQGMO_LOGICAL_ORDER, se establece la posición del cursor dentro de la secuencia lógica.

Si en este momento el primer elemento del grupo no está presente, se considera que el grupo en el que se encuentra no forma parte de la secuencia lógica.

Una vez que el cursor de examen está dentro de un grupo, puede continuar dentro del mismo grupo, incluso si se elimina el primer mensaje. Pero inicialmente nunca puede pasar a un grupo utilizando MQGMO_LOGICAL_ORDER cuando el primer elemento no está presente.

MQPMO_LOGICAL_ORDER

La opción MQPMO indica al gestor de colas cómo la aplicación coloca mensajes en grupos y segmentos de mensajes lógicos. Sólo se puede especificar en la llamada MQPUT; no es válida en la llamada MQPUT1.

Si se especifica MQPMO_LOGICAL_ORDER, indica que la aplicación utiliza llamadas MQPUT sucesivas para:

1. Poner los segmentos de cada mensaje lógico en el orden de desplazamiento de segmento creciente, empezando desde 0, sin huecos.
2. Poner todos los segmentos en un mensaje lógico antes de poner los segmentos en el siguiente mensaje lógico.
3. Poner los mensajes lógicos de cada grupo de mensajes en el orden de número de secuencia de mensaje creciente, empezando desde 1, sin huecos. IBM MQ incrementa el número de secuencia de mensaje automáticamente.
4. Poner todos los mensajes lógicos en un grupo de mensajes antes de poner los mensajes lógicos en el siguiente grupo de mensajes.

Debido a que la aplicación ha indicado al gestor de colas cómo coloca mensajes en grupos y segmentos de mensajes lógicos, la aplicación no necesita mantener y actualizar la información de grupo y de segmento referente a cada llamada MQPUT, pues el gestor de colas mantiene y actualiza esta información. En concreto, significa que la aplicación no necesita establecer los campos *GroupId*,

*MsgSeqNumber*y *Offset* en MQMD, porque el gestor de colas establece estos campos en los valores adecuados. La aplicación sólo debe establecer el campo *MsgFlags* en MQMD, para indicar cuándo los mensajes pertenecen a grupos o son segmentos de mensajes lógicos, y para indicar el último mensaje de un grupo o último segmento de un mensaje lógico.

Una vez iniciado un grupo de mensajes o un mensaje lógico, las llamadas MQPUT posteriores deben especificar los distintivos MQMF_* apropiados en *MsgFlags* en MQMD. Si la aplicación intenta colocar un mensaje que no está en un grupo cuando hay un grupo de mensajes sin terminar, o bien coloca un mensaje que no es un segmento cuando hay un mensaje lógico sin terminar, la llamada falla y devuelve el código de razón MQRC_INCOMPLETE_GROUP o MQRC_INCOMPLETE_MSG, según corresponda. Sin embargo, el gestor de colas retiene la información sobre el grupo de mensajes actual o mensaje lógico actual, y la aplicación puede terminarlos enviando un mensaje (posiblemente sin datos de mensaje de aplicación) especificando MQMF_LAST_MSG_IN_GROUP o MQMF_LAST_SEGMENT según corresponda, antes de emitir de nuevo la llamada MQPUT para colocar el mensaje que no está en el grupo o no es un segmento.

Figura 62 en la página 796 muestra las combinaciones de opciones y distintivos que son válidas y los valores de los campos *GroupId*, *MsgSeqNumber* y *Offset* que utiliza el gestor de colas en cada caso. Las combinaciones de opciones y distintivos que no aparecen en la tabla no son válidas. Las columnas de la tabla tienen los significados siguientes. "Cualquiera de los dos" significa Sí o No:

LOG ORD
Indica si la opción MQPMO_LOGICAL_ORDER se ha especificado en la llamada.

MIG
Indica si la opción MQMF_MSG_IN_GROUP o MQMF_LAST_MSG_IN_GROUP se ha especificado en la llamada.

SEG
Indica si la opción MQMF_SEGMENT o MQMF_LAST_SEGMENT se ha especificado en la llamada.

SEG OK
Indica si la opción MQMF_SEGMENTATION_ALLOWED se ha especificado en la llamada.

Cur grp
Indica si existe un grupo de mensajes actual antes de la llamada.

Cur log msg
Indica si existe un mensaje lógico actual antes de la llamada.

Otras columnas
Muestra los valores que utiliza el gestor de colas. "Anterior" denota el valor utilizado para el campo en el mensaje anterior para el descriptor de contexto de cola.

Tabla 116. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos

Opciones especificadas	Opciones especificadas	Opciones especificadas	Opciones especificadas	Estado de grupo y mensaje lógico antes de la llamada	Estado de grupo y mensaje lógico antes de la llamada	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	**GroupI**d	**MsgSeqNumber**	**Offset**
Sí	No	No	No	No	No	MQGI_NONE	1	0
Sí	No	No	Sí	No	No	Nuevo ID de grupo	1	0

Tabla 116. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos (continuación)

Opciones especificadas	Opciones especificadas	Opciones especificadas	Opciones especificadas	Estado de grupo y mensaje lógico antes de la llamada	Estado de grupo y mensaje lógico antes de la llamada	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas
Sí	No	Sí	Cualquiera de los dos	No	No	Nuevo ID de grupo	1	0
Sí	No	Sí	Cualquiera de los dos	No	Sí	ID de grupo anterior	1	Desplazamiento anterior + longitud de segmento anterior
Sí	Sí	Cualquiera de los dos	Cualquiera de los dos	No	No	Nuevo ID de grupo	1	0
Sí	Sí	Cualquiera de los dos	Cualquiera de los dos	Sí	No	ID de grupo anterior	Número de secuencia anterior + 1	0
Sí	Sí	Sí	Cualquiera de los dos	Sí	Sí	ID de grupo anterior	Número de secuencia anterior	Desplazamiento anterior + longitud de segmento anterior
No	No	No	No	Cualquiera de los dos	Cualquiera de los dos	MQGI_NONE	1	0
No	No	No	Sí	Cualquiera de los dos	Cualquiera de los dos	ID de grupo nuevo si MQGI_NONE, en otro caso, valor en campo	1	0
No	No	Sí	Cualquiera de los dos	Cualquiera de los dos	Cualquiera de los dos	ID de grupo nuevo si MQGI_NONE, en otro caso, valor en campo	1	Valor en campo
No	Sí	No	Cualquiera de los dos	Cualquiera de los dos	Cualquiera de los dos	ID de grupo nuevo si MQGI_NONE, en otro caso, valor en campo	Valor en campo	0

Tabla 116. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos (continuación)

Opciones especificadas	Opciones especificadas	Opciones especificadas	Opciones especificadas	Estado de grupo y mensaje lógico antes de la llamada	Estado de grupo y mensaje lógico antes de la llamada	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas	Valores que utiliza el gestor de colas
No	Sí	Sí	Cualquiera de los dos	Cualquiera de los dos	Cualquiera de los dos	ID de grupo nuevo si MQGI_NONE, en otro caso, valor en campo	Valor en campo	Valor en campo

Nota:

- MQPMO_LOGICAL_ORDER no es válido en la llamada MQPUT1.
- Para el campo *MsgId*, el gestor de colas genera un nuevo identificador de mensaje si se especifica MQPMO_NEW_MSG_ID o MQMI_NONE; en otro caso, utiliza el valor contenido en el campo.
- Para el campo *CorrelId*, el gestor de colas genera un nuevo identificador de correlación si se especifica MQPMO_NEW_CORREL_ID; en otro caso, utiliza el valor contenido en el campo.

Cuando se especifica MQPMO_LOGICAL_ORDER, el gestor de colas requiere que todos los mensajes de un grupo y segmentos de un mensaje lógico se coloquen con el mismo valor en el campo *Persistence* de MQMD, es decir, todos deben ser persistentes o todos deben ser no persistentes. Si esta condición no se cumple, la llamada MQPUT falla y devuelve el código de razón MQRC_INCONSISTENT_PERSISTENCE.

La opción MQPMO_LOGICAL_ORDER afecta a las unidades de trabajo de la manera siguiente:

- Si el primer mensaje físico de un grupo o mensaje lógico se coloca dentro de una unidad de trabajo, todos los demás mensajes físicos del grupo o mensaje lógico se deben colocar dentro de una unidad de trabajo, si se utiliza el mismo descriptor de contexto de cola. Sin embargo, no es necesario colocarlos dentro de la misma unidad de trabajo, lo que permite que un grupo de mensajes o un mensaje lógico que conste de muchos mensajes físicos se pueda repartir entre dos o más unidades de trabajo consecutivas para el descriptor de contexto de cola.
- Si el primer mensaje físico de un grupo o mensaje lógico no se coloca dentro de una unidad de trabajo, ninguno de los demás mensajes físicos del grupo o mensaje lógico se puede colocar dentro de una unidad de trabajo, si se utiliza el mismo descriptor de contexto de cola.

Si estas condiciones no se cumple, la llamada MQPUT falla y devuelve el código de razón MQRC_INCONSISTENT_UOW.

Cuando se especifica MQPMO_LOGICAL_ORDER, el MQMD que se proporciona en la llamada MQPUT no debe ser menor que MQMD_VERSION_2. Si esta condición no se cumple, la llamada falla y devuelve el código de razón MQRC_WRONG_MD_VERSION.

Si MQPMO_LOGICAL_ORDER no se especifica, los mensajes de grupos y los segmentos de mensajes lógicos se pueden colocar en cualquier orden, y no es necesario colocar grupos de mensajes completos o mensajes lógicos completos. Corresponde a la aplicación asegurarse de que los campos *GroupId*, *MsgSeqNumber*, *Offset* y *MsgFlags* tengan valores apropiados.

Utilice esta técnica para reiniciar un grupo de mensajes o mensaje lógico situado en el medio, después de producirse un error del sistema. Cuando se reinicia el sistema, la aplicación puede establecer los campos *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags* y *Persistence* en los valores

apropiados y luego emitir la llamada MQPUT con MQPMO_SYNCPOINT o MQPMO_NO_SYNCPOINT establecido según convenga, pero sin especificar MQPMO_LOGICAL_ORDER. Si esta llamada es satisfactoria, el gestor de colas conserva la información de grupo y segmento, y las llamadas MQPUT posteriores que utilizan ese descriptor de contexto de cola pueden especificar MQPMO_LOGICAL_ORDER en la forma habitual.

La información de grupo y segmento que el gestor de colas retiene para la llamada MQPUT está separada de la información de grupo y segmento que retiene para la llamada MQGET.

Para cualquier descriptor de contexto de cola determinado, la aplicación puede combinar llamadas MQPUT que especifican MQPMO_LOGICAL_ORDER con llamadas MQPUT que no lo hacen, pero tenga en cuenta los puntos siguientes:

- Si no se especifica MQPMO_LOGICAL_ORDER, cada llamada MQPUT satisfactoria hace que el gestor de colas establezca la información de grupo y de segmento para el descriptor de cola en los valores especificados por la aplicación, sustituyendo la información de segmento y segmento existente retenida por el gestor de colas para el descriptor de cola.
- Si no se especifica MQPMO_LOGICAL_ORDER, la llamada no fallará si existe un grupo de mensajes o un mensaje lógico actual; la llamada podría tener éxito y devolver el código de terminación MQCC_WARNING. La Tabla 117 en la página 801 muestra los distintos casos que se pueden presentar. En estos casos, si el código de terminación no es MQCC_OK, el código de razón es uno de los siguientes (según corresponda):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW

  Nota: El gestor de colas no comprueba la información de grupo y de segmento para la llamada MQPUT1.

Tabla 117. Resultado cuando la llamada MQPUT o MQCLOSE no es coherente con la información de grupo y segmento

La llamada actual es	La llamada anterior era MQPUT con MQPMO_LOGICAL_ORDER	La llamada anterior era MQPUT sin MQPMO_LOGICAL_ORDER
MQPUT con MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT sin MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE con un grupo o mensaje lógico sin terminar	MQCC_WARNING	MQCC_OK

Para las aplicaciones que colocan mensajes y segmentos en orden lógico, especifique MQPMO_LOGICAL_ORDER, pues la opción más sencilla de utilizar. Esta opción hace que la aplicación no tenga que gestionar la información de grupo y segmento, pues el gestor de colas lo hace en su lugar. Sin embargo, es posible que las aplicaciones especializadas necesiten más control que el proporcionado por la opción MQPMO_LOGICAL_ORDER, lo que se puede lograr no especificando esa opción; si lo hace, debe asegurarse de que los campos *GroupId*, *MsgSeqNumber*, *Offset*y *MsgFlags* en MQMD se hayan establecido correctamente, antes de cada llamada MQPUT o MQPUT1 .

Por ejemplo, una aplicación que desea reenviar los mensajes físicos que recibe, sin tener en cuenta si esos mensajes están en grupos o segmentos de mensajes lógicos, no debe especificar MQPMO_LOGICAL_ORDER, por dos razones:

- Si los mensajes se recuperan y se ponen en orden, la especificación MQPMO_LOGICAL_ORDER asigna un nuevo identificador de grupo a los mensajes, lo que puede hacer que sea difícil o

imposible que el emisor de los mensajes correlacione cualquier mensaje de respuesta o informe que resulte del grupo de mensajes.

- En una red compleja con múltiples rutas entre los gestores de colas de emisión y recepción, los mensajes físicos pueden llegar fuera de secuencia. Si no se especifica MQPMO_LOGICAL_ORDER ni MQGMO_LOGICAL_ORDER en la llamada MQGET, la aplicación de reenvío puede recuperar y reenviar cada mensaje físico tan pronto como llegue, sin esperar a que llegue el mensaje siguiente en orden lógico.

Las aplicaciones que generan mensajes de informe para mensajes de grupos o segmentos de mensajes lógicos también no deben especificar MQPMO_LOGICAL_ORDER al colocar el mensaje de informe.

MQPMO_LOGICAL_ORDER se puede especificar con cualquiera de las demás opciones MQPMO_*.

Colocación de grupos ordenados lógicamente en una cola de clúster (MQOO_BIND_ON_GROUP)

La opción MQOO_BIND_ON_OPEN garantiza que todos los mensajes de la aplicación, y por lo tanto todos los grupos, se envíen a una misma instancia. Esto tiene el inconveniente de que el tráfico de la aplicación no está equilibrado entre las diversas de una cola de clúster. Para habilitar el equilibrado de la carga y al mismo tiempo mantener intactos los grupos de mensajes, debe establecer las opciones siguientes:

- La llamada MQPUT debe especificar MQPMO_LOGICAL_ORDER
- La llamada MQOPEN debe especificar una de las dos opciones siguientes:
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF, y la definición de cola debe especificar DEFBIND(GROUP)

Entonces el equilibrio de la carga de trabajo se activa *entre grupos* de mensajes, sin necesidad de ejecutar MQCLOSE y MQOPEN para la cola. *Entre grupos* significa que MQMF_MSG_IN_GROUP se establece en MQMD(v2) o MQMDE, y no hay ningún grupo parcialmente completo en curso. Cuando un grupo está en curso, se reutilizan el gestor de colas resuelto y el nombre de cola en el descriptor de contexto de objeto.

Si el mensaje anterior era MQPMO_LOGICAL_ORDER o se ha establecido MQMF_MSG_IN_GROUP, pero el mensaje actual no forma parte del grupo, la llamada PUT falla y devuelve MQRC_INCOMPLETE_GROUP.

Si una operación MQPUT individual no especifica MQPMO_LOGICAL_ORDER, y no hay ningún grupo actual activo, el equilibrado de la carga de trabajo se activa para ese mensaje (como si la llamada MQOPEN hubiera especificado MQOO_BIND_NOT_FIXED).

No se lleva a cabo ninguna reasignación para los mensajes enlazados a un destino utilizando MQOO_BIND_ON_GROUP. Para obtener más información sobre la reasignación, consulte “Grupos de mensajes” en la página 45.

Agrupación de mensajes lógicos

Hay dos razones principales para utilizar mensajes lógicos en un grupo:

- Puede que haya que procesar los mensajes en un orden determinado.
- Puede que haya que procesar cada mensaje de un grupo de una forma relacionada.

En cualquier caso, recupere el grupo completo con la misma instancia de aplicación obtenedora.

Por ejemplo, suponga que el grupo consta de cuatro mensajes lógicos. La aplicación colocadora tiene este aspecto:

```
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
```

La aplicación obtenedora especifica la opción MQGMO_ALL_MSGS_AVAILABLE para el primer mensaje en el grupo. Esto garantiza que el proceso no se inicie hasta que hayan llegado todos los mensajes del grupo. La opción MQGMO_ALL_MSGS_AVAILABLE se pasa por alto en los siguientes mensajes del grupo.

Cuando se recupera el primer mensaje lógico del grupo, se puede utilizar MQGMO_LOGICAL_ORDER para asegurarse de que los mensajes lógicos restantes del grupo se recuperen en orden.

Por lo tanto, la aplicación obtenedora será algo parecido a esto:

```
/* 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
```

Para obtener más ejemplos de agrupación de mensajes, consulte “Segmentación de aplicación de un mensaje lógico” en la página 815 y “Colocación y obtención de un grupo que abarca varias unidades de trabajo” en la página 803.

Atención: Cuando se utiliza la publicación/suscripción para enviar mensajes a un tema (o colocar mensajes en un alias de tema), no se permite la agrupación y segmentación de mensajes.

Puesto que las suscripciones se pueden crear y eliminar independientemente de la actividad de publicación, no se puede garantizar que un suscriptor reciba un grupo de mensajes completo o todos los segmentos de un mensaje; consulte RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Para obtener información sobre cómo permitir que una aplicación solicite que a un grupo de mensajes se le asigne a la misma instancia de destino en colas de clúster, consulte DefBind.

Colocación y obtención de un grupo que abarca varias unidades de trabajo

En el caso anterior, los mensajes o segmentos no pueden iniciarse para dejar el nodo (si su destino es remoto) ni iniciarse para ser recuperados mientras no se haya colocado el grupo entero y se haya confirmado la unidad de trabajo. Puede que esto no sea lo que desea si se tarda mucho tiempo en colocar todo el grupo o si el espacio de cola está limitado en el nodo. Para evitar esto, coloque el grupo en varias unidades de trabajo.

Si el grupo se coloca en varias unidades de trabajo, es posible que parte del grupo se confirme aun cuando falle la aplicación colocadora. Por lo tanto, la aplicación tiene que guardar información de estado, confirmada con cada unidad de trabajo, que puede utilizar tras un reinicio para reanudar un grupo incompleto. El lugar más sencillo para registrar esta información es una cola STATUS. Si se ha colocado satisfactoriamente un grupo completo, la cola STATUS estará vacía.

Si hay una segmentación implicada, la lógica es similar. En tal caso, **StatusInfo** tiene que incluir el *Offset*.

A continuación se muestra un ejemplo de cómo colocar el grupo en varias unidades de trabajo:

```
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
```

Si todas las unidades de trabajo se han confirmado, el grupo completo se ha colocado correctamente y la cola STATUS estará vacía. En caso contrario, habrá que reanudar el grupo en el punto indicado por la información de estado. MQPMO_LOGICAL_ORDER no se puede utilizar en la primera colocación, pero después sí se podrá.

El proceso de reinicio es similar al siguiente:

```
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
```

En la aplicación de obtención, puede que desee empezar a procesar los mensajes de un grupo antes de que haya llegado el grupo entero. Esto mejora los tiempos de respuesta en los mensajes del grupo y también significa que no se necesita almacenamiento para todo el grupo. Para poder aprovechar los beneficios, utilice varias unidades de trabajo por cada grupo de mensajes. Por razones de recuperación, hay que recuperar cada uno de los mensajes dentro de una unidad de trabajo.

En cuanto a la correspondiente aplicación colocadora, esto requiere registrar automáticamente la información de estado en algún lugar a medida que se confirma cada unidad de trabajo. También en este caso, el lugar más sencillo donde registrar esta información es la cola STATUS. Si se ha procesado satisfactoriamente un grupo completo, la cola STATUS estará vacía.

Nota: En las unidades de trabajo intermedias, se pueden evitar las llamadas MQGET de la cola STATUS especificando que cada MQPUT a la cola de estado sea un segmento de mensaje (es decir, estableciendo el distintivo MQMF_SEGMENT), en lugar de colocar un mensaje nuevo completo por cada unidad de trabajo. En la última unidad de trabajo, se coloca un segmento final en la cola de estado especificando MQMF_LAST_SEGMENT y luego se borra la información de estado con un MQGET que especifique MQGMO_COMPLETE_MSG.

Durante el proceso de reinicio, en lugar de utilizar un único MQGET para obtener un posible mensaje de estado, examine la cola de estado con MQGMO_LOGICAL_ORDER hasta alcanzar el último segmento (es decir, hasta que no se devuelvan más segmentos). En la primera unidad de trabajo después del reinicio, especifique también el desplazamiento de forma explícita al colocar el segmento de estado.

En el ejemplo siguiente, solo se tienen en cuenta los mensajes dentro de un grupo, suponiendo que el búfer de la aplicación sea siempre lo suficientemente grande como para dar cabida al mensaje completo, tanto si está segmentado como si no. Por tanto, MQGMO_COMPLETE_MSG se especifica en cada MQGET. Se aplican los mismos principios si hay una segmentación implicada (en tal caso, StatusInfo tiene que incluir el *Offset*).

Para simplificar, suponemos que se recupera un máximo de 4 mensajes en una única unidad de trabajo:

```
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
```

Si todas las unidades de trabajo se han confirmado, el grupo completo se ha recuperado correctamente y la cola STATUS estará vacía. En caso contrario, habrá que reanudar el grupo en el punto indicado por la información de estado. MQGMO_LOGICAL_ORDER no se puede utilizar en la primera recuperación, pero después sí se podrá.

El proceso de reinicio es similar al siguiente:

```
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
```

Obtener un mensaje concreto

Existen varias formas de obtener un mensaje concreto de una cola. Estos son: Seleccionar el MsgId y el CorrelId, seleccionando GroupId, MsgSeqNumber y Offset, y seleccionando MsgToken. También puede utilizar una serie de selección cuando abre la cola.

Para obtener un mensaje concreto de una cola, utilice los campos `MsgId` y `CorrelId` de la estructura MQMD. No obstante, las aplicaciones pueden establecer estos campos de forma explícita, de modo que los valores que especifique pueden no identificar un único mensaje. La Tabla 118 en la página 806 muestra qué mensaje se recupera para los valores posibles de estos campos. Estos campos se omiten en la entrada si especifica MQGMO_MSG_UNDER_CURSOR en el parámetro **GetMsgOpts** de la llamada MQGET.

Tabla 118. Utilización de identificadores de mensajes y correlación

Para recuperar...	**MsgId**	**CorrelId**
Primer mensaje de la cola	`MQMI_NONE`	`MQCI_NONE`
Primer mensaje que coincide con *`MsgId`*	No de cero	`MQCI_NONE`
Primer mensaje que coincide con *`CorrelId`*	`MQMI_NONE`	No de cero
Primer mensaje que coincide con *`MsgId`* y *`CorrelId`*	No de cero	No de cero

En cada caso, en *primer* lugar significa el primer mensaje que cumple con el criterio de selección, a menos que se haya especificado MQGMO_BROWSE_NEXT is, cuando significa el *siguiente* mensaje de la secuencia que cumple con el criterio de selección.

En la devolución, la llamada MQGET establece los campos *`MsgId`* y *`CorrelId`* en los identificadores de mensaje y correlación del mensaje devuelto, si los hay.

Si establece el campo *`Version`* de la estructura MQMD en 2, puede utilizar los campos *`GroupId`*, *`MsgSeqNumber`* y *`Offset`*. La Tabla 119 en la página 806 muestra qué mensaje se recupera para los valores posibles de estos campos.

Tabla 119. Utilización del identificador de grupo

Para recuperar...	**Opciones de coincidencia**
Primer mensaje de la cola	MQMO_NONE
Primer mensaje que coincide con `MsgId`	MQMO_MATCH_MSG_ID
Primer mensaje que coincide con `CorrelId`	MQMO_MATCH_CORREL_ID
Primer mensaje que coincide con `GroupId`	MQMO_MATCH_GROUP_ID
Primer mensaje que coincide con `MsgSeqNumber`	MQMO_MATCH_MSG_SEQ_NUMBER
Primer mensaje que coincide con `MsgToken`	MQMO_MATCH_MSG_TOKEN
Primer mensaje que coincide con `Offset`	MQMO_MATCH_OFFSET

Notas:

1. MQMO_MATCH_XXX implica que el campo *XXX* de la estructura MQMD se establece en el valor para el que se ha de encontrar la coincidencia.
2. Los distintivos MQMO se pueden combinar. Por ejemplo, MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER y MQMO_MATCH_OFFSET se pueden utilizar conjuntamente para proporcionar el segmento identificado mediante los campos `GroupId`, `MsgSeqNumber` y `Offset`.

3. Si especifica MQGMO_LOGICAL_ORDER, el mensaje que está intentando recuperar se verá afectado debido a que la opción depende de la información controlada por el manejador de cola. Para obtener más información, consulte la sección "Ordenación lógica y física" en la página 794 y la sección Opciones.

La llamada MQGET suele recuperar el primer mensaje de una cola. Si especifica un mensaje concreto cuando utiliza la llamada MQGET, el gestor de colas debe buscar la cola hasta que encuentra dicho mensaje. Esto puede afecta al rendimiento de su aplicación.

Si está utilizando la Versión 2 o posterior de la estructura MQGMO y no especifica los distintivos MQMO_MATCH_MSG_ID o MQMO_MATCH_CORREL_ID, no es necesario que restablezca los campos `MsgId` o `CorrelId` entre llamadas MQGET.

z/OS En IBM MQ for z/OS, se puede utilizar el atributo `IndexType` para aumentar la velocidad de las operaciones MQGET en la cola. Para obtener más información, consulte "Tipo de índice" en la página 811.

Puede obtener un mensaje específico de una cola especificando su MsgToken y MatchOption MQMO_MATCH_MSG_TOKEN en la estructura MQGMO. El MsgToken lo devuelve la llamada MQPUT que originalmente ha colocado el mensaje en la cola o las operaciones MQGET anteriores, y permanece constante a menos que se reinicie el gestor de colas.

Si solo está interesado en un subconjunto de mensajes de la cola, puede especificar qué mensajes desea procesar utilizando una serie de selección con la llamada MQOPEN o MQSUB. A continuación, MQGET recupera el siguiente mensaje que cumple con dicha serie de selección. Para obtener más información acerca de las series de selección, consulte la sección "Selectores" en la página 31.

Mejora del rendimiento de mensajes no persistentes

Cuando un cliente necesita un mensaje de un servidor, envía una petición al servidor. Envía una petición separada para cada uno de los mensajes que consume. Para mejorar el rendimiento de un cliente que consume mensajes no persistentes evitando tener que enviar estos mensajes de petición, un cliente se puede configurar para que utilice *lectura anticipada*. La lectura anticipada permite enviar mensajes a un cliente sin que una aplicación tenga que solicitarlos.

Cuando la lectura anticipada está habilitada, se envían mensajes a un almacenamiento intermedio de memoria interna en el cliente, llamado *almacenamiento intermedio de lectura anticipada*. El cliente tendrá un almacenamiento intermedio de lectura anticipada para cada cola que tenga abierta con lectura anticipada habilitada. Los mensajes del almacenamiento intermedio de lectura anticipada no tienen persistencia. El cliente actualiza periódicamente el servidor con información sobre la cantidad de datos que ha consumido.

Cuando se llama a MQOPEN con MQOO_READ_AHEAD, el cliente de IBM MQ sólo permite la lectura anticipada si se cumplen ciertas condiciones. Estas condiciones incluyen:

- La aplicación cliente debe compilarse y enlazarse a las bibliotecas de cliente MQI de IBM MQ con hebras.
- El canal de cliente debe utilizar el protocolo TCP/IP
- El canal debe tener un valor SharingConversations (SHARECNV) distinto de cero tanto en las definiciones de canal de cliente y servidor.

La utilización de la lectura anticipada puede mejorar el rendimiento al consumir mensajes no persistentes de una aplicación cliente. Esta mejora en el rendimiento está disponible para las aplicaciones MQI y JMS. Las aplicaciones cliente que utilizan MQGET o consumo asíncrono se benefician de las mejoras de rendimiento al consumir mensajes no persistentes.

No todos los diseños de aplicaciones cliente son adecuados para utilizar lectura anticipada ya que no todas las opciones están soportadas para utilizarlas con lectura anticipada y algunas opciones deben ser coherentes entre llamadas MQGET cuando la lectura anticipada está habilitada. Si un cliente altera sus criterios de selección entre llamadas MQGET, los mensajes que se almacenan en el almacenamiento intermedio de lectura anticipada permanecerán abandonados en el almacenamiento intermedio de lectura anticipada del cliente.

Si ya no se necesita un registro de reserva de mensajes bloqueados con los criterios de selección anteriores, se puede establecer un intervalo de depuración configurable en el cliente para purgar automáticamente estos mensajes del cliente. El intervalo de purga es un intervalo de un grupo de opciones de ajuste de lectura anticipada determinadas por el cliente. Es posible ajustar estas opciones para cumplir con sus requisitos.

Si se reinicia una aplicación cliente, los mensajes en el almacenamiento intermedio de lectura anticipada se pueden perder. Por el contrario, un mensaje que se ha movido a un almacenamiento intermedio de lectura anticipada se puede suprimir de la cola subyacente; esto no provoca que se elimine del almacenamiento intermedio, de modo que una llamada MQGET que utilice la lectura anticipada puede devolver un mensaje que ya no existe.

La lectura anticipada sólo se lleva a cabo para los enlaces de cliente. El atributo se ignora para el resto de los enlaces.

La lectura anticipada no afecta a los desencadenantes. No se genera ningún mensaje desencadenante cuando el cliente lee un mensaje de forma anticipada. La lectura anticipada no genera información de contabilidad ni estadística cuando está habilitada.

Uso de la lectura anticipada con la mensajería de suscripción de publicación

Cuando una aplicación de suscripción especifica una cola de destino a la que se envían las publicaciones, el valor DEFREADA de la cola especificada se utiliza como el valor predeterminado de lectura anticipada.

Cuando una aplicación de suscripción solicita que IBM MQ gestione el destino al que se envían las publicaciones, se crea una cola gestionada como una cola dinámica basada en una cola de modelo predefinida. Como valor de lectura anticipada predeterminada, se utiliza el valor DEFREADA de la cola modelo. Se utilizan las colas de modelos predeterminados SYSTEM.DURABLE.PUBLICATIONS.MODEL o SYSTEM.NONDURABLE.PUBLICATIONS.MODEL, a menos que se defina una cola modelo para este tema o un tema padre.

Conceptos relacionados

“Ajuste del rendimiento de los mensajes no persistentes en AIX” en la página 810
Si utiliza AIX V5.3 o posterior, se recomienda establecer el parámetro de ajuste para utilizar el rendimiento completo de los mensajes no persistentes.

Tareas relacionadas

“Habilitación e inhabilitación de la lectura anticipada” en la página 810
De forma predeterminada, la lectura anticipada está inhabilitada. Se puede habilitar la lectura anticipada a nivel de cola o de aplicación.

Referencia relacionada

“Opciones de MQGET y lectura anticipada” en la página 808
No todas las opciones de MQGET se pueden utilizar cuando se habilita la lectura anticipada; algunas opciones son necesarias para asegurar la coherencia entre llamadas MQGET.

Opciones de MQGET y lectura anticipada

No todas las opciones de MQGET se pueden utilizar cuando se habilita la lectura anticipada; algunas opciones son necesarias para asegurar la coherencia entre llamadas MQGET.

Cuando se llama a MQOPEN con MQOO_READ_AHEAD, el cliente de IBM MQ sólo permite la lectura anticipada si se cumplen ciertas condiciones. Estas condiciones incluyen:

- La aplicación cliente debe compilarse y enlazarse a las bibliotecas de cliente MQI de IBM MQ con hebras.
- El canal de cliente debe utilizar el protocolo TCP/IP
- El canal debe tener un valor SharingConversations (SHARECNV) distinto de cero tanto en las definiciones de canal de cliente y servidor.

La tabla siguiente indica las opciones que se pueden utilizar con la lectura anticipada y si se pueden modificar entre llamadas MQGET.

Tabla 120. Opciones de MQGET y lectura anticipada			
Valores y opciones de MQGET	**Opciones permitidas cuando la lectura anticipada está habilitada y se pueden alterar entre llamadas MQGET [5]**	**Permitidas cuando la lectura anticipada está habilitada, pero no se pueden modificar entre llamadas MQGET[1]**	**Opciones de MQGET que no están permitidas cuando la lectura anticipada está habilitada [2]**
Valores de MQGET MQMD	MsgId[3] CorrelId [3]	Encoding CodedCharSetId	
Opciones de MQGET MQGMO	• 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

Notas:

1. Si estas opciones se alteran entre llamadas MQGET, se devuelve un código de razón MQRC_OPTIONS_CHANGED.
2. Si estas opciones se especifican en la primera llamada MQGET, la lectura anticipada está inhabilitada. Si estas opciones se especifican en una llamada MQGET posterior, se devuelve el código de razón MQRC_OPTIONS_ERROR.
3. Si una aplicación cliente altera los valores MsgId y CorrelId entre llamadas MQGET, es posible que los mensajes con los valores anteriores ya se hayan enviado al cliente y que permanezcan en el almacenamiento intermedio de lectura anticipada del cliente hasta que se consuman (o se depuren automáticamente).
4. MQGMO_MSG_UNDER_CURSOR no es posible con la lectura anticipada. La lectura anticipada está inhabilitada cuando se especifican MQOO_BROWSE y una de las opciones MQOO_INPUT_SHARED o MQOO_INPUT_EXCLUSIVE cuando se abre la cola.
5. Cuando la lectura anticipada está habilitada, la primera operación MQGET determina si los mensajes se deben examinar u obtener de una cola. Si la aplicación cliente utiliza entonces MQGET con opciones modificadas, tal como intentar examinar después de una operación Get inicial, o intentar obtener después de un Browse inicial, se devuelve un código de razón MQRC_OPTIONS_CHANGED.

Si un cliente modifica sus criterios de selección entre llamadas MQGET, los mensajes que se almacenan en el almacenamiento intermedio de lectura anticipada que coinciden con los criterios de selección iniciales no son consumidos por la aplicación cliente, y permanecen abandonados en el almacenamiento intermedio de lectura anticipada del cliente. En las situaciones en las que el almacenamiento intermedio de lectura anticipada del cliente contiene muchos mensajes abandonados, se pierden las ventajas de la lectura anticipada y es necesaria una solicitud separada dirigida al servidor para cada mensaje consumido. Para determinar si la lectura anticipada se está utilizando de forma eficiente, puede utilizar el parámetro de estado de conexión, READA.

La lectura anticipada se puede inhibir a petición de una aplicación debido a opciones incompatibles especificadas en la primera llamada MQGET. En esta situación, el estado de conexión muestra la lectura anticipada como inhibida.

Si, debido a estas restricciones en MQGET, decide que un diseño de aplicación cliente no es adecuado para la lectura anticipada, especifique la opción MQOO_READ_AHEAD_NO para MQOPEN. Como alternativa, establezca en NO o DISABLED el valor predeterminado de lectura anticipada para la cola que se debe abrir o modificar.

Habilitación e inhabilitación de la lectura anticipada
De forma predeterminada, la lectura anticipada está inhabilitada. Se puede habilitar la lectura anticipada a nivel de cola o de aplicación.

Acerca de esta tarea

Cuando se llama a MQOPEN con MQOO_READ_AHEAD, el cliente de IBM MQ sólo permite la lectura anticipada si se cumplen ciertas condiciones. Estas condiciones incluyen:

- La aplicación cliente debe compilarse y enlazarse a las bibliotecas de cliente MQI de IBM MQ con hebras.
- El canal de cliente debe utilizar el protocolo TCP/IP
- El canal debe tener un valor SharingConversations (SHARECNV) distinto de cero tanto en las definiciones de canal de cliente y servidor.

Para habilitar la lectura anticipada:

- Para configurar la lectura anticipada a nivel de cola, establezca el atributo de cola DEFREADA a YES.
- Para configurar la lectura anticipada a nivel de aplicación:
  - para utilizar la lectura anticipada siempre que sea posible, utilice la opción MQOO_READ_AHEAD en la llamada de función MQOPEN. No es posible que un aplicación cliente utilice la lectura anticipada si el atributo de cola DEFREADA se ha establecido a DISABLED.
  - para utilizar la lectura anticipada solo cuando esta está habilitada en una cola, utilice la opción MQOO_READ_AHEAD_AS_Q_DEF en la llamada de función MQOPEN.

Si un diseño de aplicación cliente no es adecuado para la lectura anticipada, puede inhabilitarlo:

- a nivel de cola, estableciendo el atributo de cola DEFREADA a NO si no desea que se utilice la lectura anticipada a menos que la solicite una aplicación cliente, o a DISABLED si no desea que la lectura anticipada se utilice, independientemente de si una aplicación cliente la necesita o no.
- a nivel de aplicación, utilizando la opción MQOO_NO_READ_AHEAD en la llamada de función MQOPEN.

Dos opciones MQCLOSE permiten configurar lo que sucede a cualquier mensaje que se almacena en el búfer de lectura anticipada si la cola está cerrada.

- Utilice MQCO_IMMEDIATE para descartar los mensajes del búfer de lectura anticipada.
- Utilice MQCO_QUIESCE para asegurarse de que la aplicación consuma los mensajes del búfer de lectura anticipada antes de que se cierre la cola. Cuando se emite MQCLOSE con MQCO_QUIESCE y quedan mensajes en el búfer de lectura anticipada, MQRC_READ_AHEAD_MSGS retorna MQCC_WARNING.

Ajuste del rendimiento de los mensajes no persistentes en AIX
Si utiliza AIX V5.3 o posterior, se recomienda establecer el parámetro de ajuste para utilizar el rendimiento completo de los mensajes no persistentes.

Para establecer el parámetro de ajuste para que entre en vigor inmediatamente, emita el mandato siguiente como usuario root:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Para establecer el parámetro de ajuste para que entre en vigor inmediatamente y persista después de un reinicio, emita el mandato siguiente como usuario root:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Normalmente, los mensajes no persistentes solo se mantienen en memoria, pero hay algunos casos en los que AIX puede planificar los mensajes no persistentes para que se graben en disco. Los mensajes que se planifican para grabarse en disco no pueden obtenerse mediante MQGET hasta que finalice la grabación en disco. El mandato de ajuste recomendado modifica este umbral. En lugar de planificar que se graben en disco los mensajes cuando hay 16 kilobytes de datos en cola, la grabación en disco se

realiza solamente cuando el almacenamiento real de la máquina está casi lleno. Esta es una alteración global y puede afectar a otros componentes de software.

En AIX, cuando se utilizan aplicaciones multihebra y especialmente cuando se ejecutan en máquinas con varios procesadores, se recomienda encarecidamente establecer AIXTHREAD_SCOPE=S en el ID de mqm `.profile` o establecer AIXTHREAD_SCOPE=S en el entorno antes de iniciar la aplicación, para obtener un mejor rendimiento y una planificación más sólida. Por ejemplo:

```
export AIXTHREAD_SCOPE=S
```

El establecimiento de AIXTHREAD_SCOPE=S significa que las hebras de usuario creadas con atributos predeterminados se colocan en el ámbito de contención de todo el sistema. Si una hebra de usuario se crea con ámbito de contención de todo el sistema, se enlaza con una hebra de kernel y el kernel la planifica. La hebra de kernel subyacente no se comparte con ninguna otra hebra de usuario.

Descriptores de archivo

Al ejecutar un proceso multihebra, como el proceso de agente, es posible que alcance el límite flexible para los descriptores de archivo. Este límite proporciona el código de razón de IBM MQ `MQRC_UNEXPECTED_ERROR (2195)` y, si hay suficientes descriptores de archivo, un archivo de IBM MQ FFST™.

Para evitar este problema, puede aumentar el límite de procesos para el número de descriptores de archivo. Para ello, cambie el atributo `nofiles` en `/etc/security/limits` a 10.000 para el ID de usuario mqm o en la stanza predeterminada.

Límites de recursos del sistema

Establezca el límite de recursos del sistema para segmentos de datos y segmentos de pilas en ilimitado utilizando los siguientes mandatos en un indicador de mandatos:

```
ulimit -d unlimited
ulimit -s unlimited
```

Tipo de índice

El atributo de cola, *IndexType*, especifica el tipo de índice que el gestor de colas mantiene para aumentar la velocidad de las operaciones MQGET en la cola.

Nota: Solo está soportado en IBM MQ for z/OS.

Tiene cinco opciones:

Valor	Descripción
NINGUNO	No se mantiene ningún índice. Utilice este valor cuando recupere los mensajes secuencialmente (consulte “Prioridad” en la página 793).
GROUPID	Se mantiene un índice de identificadores de grupo. Debe utilizar este tipo de índice si desea una ordenación lógica de los grupos de mensajes (consulte “Ordenación lógica y física” en la página 794).
MSGID	Se mantiene un índice de los identificadores de mensaje. Utilice este valor cuando recupere mensajes utilizando el campo *MsgId* como criterio de selección en la llamada MQGET (consulte “Obtener un mensaje concreto” en la página 806).
MSGTOKEN	Se mantiene un índice de las señales de mensaje.
CORRELID	Se mantiene un índice de identificadores de correlación. Utilice este valor cuando recupere mensajes utilizando el campo *CorrelId* como criterio de selección en la llamada MQGET (consulte “Obtener un mensaje concreto” en la página 806).

Nota:

1. Si está realiza la indexación utilizando la opción MSGID o la opción CORRELID, establezca los parámetros **MsgId** o **CorrelId** relativos en MQMD. No se recomienda establecer ambos.
2. Para examinar, se utiliza el mecanismo de índice para encontrar un mensaje si una cola coincide con todas las condiciones siguientes:
   - Tiene el tipo de índice MSGID, CORRELID o GROUPID
   - Se examina con el mismo tipo de ID
   - Tiene mensajes de solo una prioridad
3. Evite las colas (indexadas por *MsgId* o *CorrelId*) que contienen miles de mensajes, porque esto afecta al tiempo de reinicio. (Esto no se aplica a los mensajes no persistentes, ya que se suprimen durante el reinicio).
4. MSGTOKEN se utiliza para definir las colas gestionadas por el gestor de carga de trabajo de z/OS.

Para obtener una descripción completa del atributo **IndexType**, consulte IndexType. Para obtener más información sobre el atributo **IndexType**, consulte "Consideraciones de diseño y rendimiento para aplicaciones de z/OS" en la página 67.

Manejo de mensajes de más de 4 MB de tamaño

Los mensajes pueden ser demasiado grandes para la aplicación, la cola o el gestor de colas. En función del entorno, IBM MQ proporciona una serie de formas de tratar los mensajes que tienen más de 4 MB.

Puede aumentar el atributo **MaxMsgLength** hasta 100 MB en todos los sistemas IBM MQ en V6 o posteriores. Establezca este valor para que refleje el tamaño de los mensajes que usan la cola. En sistemas IBM MQ distintos de IBM MQ for z/OS, también puede:

1. Usar mensajes segmentados. (Los mensajes los puede segmentar la aplicación o el gestor de colas.)
2. Usar mensajes de referencia.

Cada uno de estos enfoques se describe en el resto de esta sección.

Aumentar la longitud máxima del mensaje

El atributo del gestor de colas **MaxMsgLength** define la longitud máxima de un mensaje que un gestor de colas puede manejar. Del mismo modo, el atributo de cola **MaxMsgLength** es la longitud máxima de un mensaje que una cola pueda manejar. La longitud de mensaje máxima predeterminada admitida depende del entorno en el que esté trabajando.

Si está manejando mensajes de gran tamaño, puede modificar estos atributos de forma independiente en plataformas que no sean z/OS. Puede establecer el valor de atributo del gestor de colas en el rango de 32768 bytes a 100 MB.

Atención: En IBM MQ for z/OS el atributo **MaxMsgLength** del gestor de colas está codificado en 100 MB.

En todas las plataformas, puede establecer el valor de atributo de cola en el rango de 0 a 100 MB.

Después de cambiar uno o dos de los atributos **MaxMsgLength**, reinicie las aplicaciones y los canales para asegurarse de que los cambios entren en vigor.

Cuando se realizan estos cambios, la longitud del mensaje debe ser inferior o igual a la cola y los atributos **MaxMsgLength** del gestor de colas. Sin embargo, los mensajes existentes pueden ser más largos que cualquiera de los dos atributos.

Si el mensaje es demasiado grande para la cola, se devuelve MQRC_MSG_TOO_BIG_FOR_Q.
De la misma forma, si el mensaje es demasiado grande para el gestor de cola, se devuelve MQRC_MSG_TOO_BIG_FOR_Q_MGR.

Este método de manejo de mensajes de gran tamaño es fácil y cómodo. Sin embargo, tenga en cuenta los siguientes factores antes de usarlo:

- La uniformidad entre los gestores de colas se reduce. El tamaño máximo de los datos de mensaje viene determinado por el *MaxMsgLength* para cada cola (incluidas las colas de transmisión) en la que se colocará el mensaje. A menudo, este valor se toma de forma predeterminada en el *MaxMsgLength* del gestor de colas, especialmente para las colas de transmisión. Esto hace que sea difícil predecir si un mensaje es demasiado grande cuando se trata del traslado a un gestor de colas remoto.
- Se aumenta el uso de los recursos del sistema. Por ejemplo, las aplicaciones necesitan almacenamientos intermedios más grandes, y en algunas plataformas, es posible que se aumente el uso del almacenamiento compartido. El almacenamiento de cola sólo debe verse afectado si es necesario para los mensajes más grandes.
- El proceso por lotes de canal se ve afectado. Un mensaje grande sigue contando como sólo un mensaje en cuanto al recuento de lotes, pero necesita más tiempo para transmitir, incrementando de este modo los tiempos de respuesta para otros mensajes.

Multi *Segmentación de mensajes*

Utilice esta información para obtener información acerca de la segmentación de mensajes. Esta característica no se admite en IBM MQ for z/OS ni mediante aplicaciones que utilizan IBM MQ classes for JMS.

El aumento de la longitud máxima del mensaje tal como se explica en el tema "Aumentar la longitud máxima del mensaje" en la página 812 tiene algunas implicaciones negativas. Además, todavía puede dar como resultado que el mensaje sea demasiado grande para la cola o el gestor de colas. En tales casos, puede segmentar un mensaje. Para obtener información sobre los segmentos, consulte "Grupos de mensajes" en la página 45.

En las secciones siguientes se observan los usos comunes para segmentar mensajes. Para colocar y obtener de forma destructiva, se supone que las llamadas MQPUT o MQGET *siempre* funcionan dentro de una unidad de trabajo. Planteese el uso de esta técnica para reducir la posibilidad de que haya grupos incompletos en la red. Se supone la confirmación de una sola fase por parte del gestor de colas, pero otras técnicas de coordinación son igualmente válidas.

Además, en las aplicaciones de obtención, se presupone que si hay varios servidores procesando la misma cola, cada servidor utiliza código parecido, para que los servidores siempre encuentren un mensaje o segmento que se espera que esté ahí (porque se ha especificado anteriormente MQGMO_ALL_MSGS_AVAILABLE o MQGMO_ALL_SEGMENTS_AVAILABLE).

Atención: Cuando se utiliza la publicación/suscripción para enviar mensajes a un tema (o colocar mensajes en un alias de tema), no se permite la agrupación y segmentación de mensajes.

Puesto que las suscripciones se pueden crear y eliminar independientemente de la actividad de publicación, no se puede garantizar que un suscriptor reciba un grupo de mensajes completo o todos los segmentos de un mensaje; consulte RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Colocación y obtención de un mensaje segmentado que abarque unidades de trabajo

Puede poner y obtener un mensaje segmentado que abarque una unidad de trabajo de forma similar a la de "Colocación y obtención de un grupo que abarca varias unidades de trabajo" en la página 803.

Sin embargo, no puede poner o recibir mensajes segmentados en una unidad de trabajo global.

Multi *Segmentación y reensamblaje realizados por el gestor de colas*

Este es el escenario más sencillo, en el que una aplicación coloca un mensaje para que otra lo recupere. El mensaje puede ser grande: no demasiado grande para que las aplicaciones colocadora u obtenedora lo puedan manejar en un único búfer, pero sí demasiado grande para el gestor de colas o la cola en la que se va a colocar el mensaje.

Los únicos cambios necesarios en estas aplicaciones consisten en que la aplicación colocadora autorice al gestor de colas a realizar una segmentación en caso de ser necesario:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

y en que la aplicación obtenedora solicite al gestor de colas que reensamble el mensaje en caso de estar segmentado:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

En este escenario, que no puede ser más sencillo, la aplicación tiene que restablecer el campo GroupId a MQGI_NONE antes de la llamada MQPUT para que el gestor de colas pueda generar un identificador de grupo exclusivo para cada mensaje. Si esto no se hace, mensajes no relacionados podrían tener el mismo identificador de grupo, lo que luego podría dar lugar a un procesamiento incorrecto.

El búfer de la aplicación tiene que ser lo suficientemente grande como para contener el mensaje reensamblado (a menos que se incluya la opción MQGMO_ACCEPT_TRUNCATED_MSG).

Si hay que modificar el atributo MAXMSGLEN de una cola para dar cabida a la segmentación de mensajes, tenga en cuenta lo siguiente:

- El segmento de mensaje mínimo soportado en una cola local es de 16 bytes.
- En una cola de transmisión, MAXMSGLEN también ha de incluir el espacio necesario para las cabeceras. Considere usar un valor de al menos 4000 bytes por encima de la longitud máxima esperada en cualquier segmento de mensaje que pueda colocarse en una cola de transmisión.

Si fuera necesaria una conversión de datos, la aplicación obtenedora podría hacerlo especificando MQGMO_CONVERT. Esto no debería plantear mayores problemas, ya que a la salida de conversión se le presenta el mensaje completo. No intente convertir los datos en un canal emisor si el mensaje está segmentado y el formato de los datos es tal que la salida de conversión de datos no pueda llevar a cabo la conversión en datos incompletos.

Multi *Segmentación de aplicaciones*

La segmentación de aplicaciones se utiliza cuando la segmentación del gestor de colas no es adecuada o cuando las aplicaciones requieren una conversión de datos con límites de segmento específicos.

La segmentación de aplicaciones se utiliza por dos razones principales:

1. La segmentación del gestor de colas por sí sola no es adecuada porque el mensaje es demasiado grande para ser manejado en un único búfer por las aplicaciones.
2. La conversión de datos tiene que ser realizada por los canales emisores y el formato es tal que la aplicación colocadora tiene que estipular dónde han de estar los límites de segmento para que se pueda convertir un segmento individual.

Sin embargo, si la conversión de datos no es un problema, o si la aplicación de obtención utiliza siempre MQGMO_COMPLETE_MSG, también se puede permitir la segmentación del gestor de colas especificando MQMF_SEGMENTATION_ALLOWED. En este ejemplo, la aplicación divide el mensaje en cuatro segmentos:

```
   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
```

Si no se usa MQPMO_LOGICAL_ORDER, la aplicación tiene que establecer el *Offset* (desplazamiento) y la longitud de cada segmento. En este caso, el estado lógico no se mantiene automáticamente.

La aplicación de obtención no puede garantizar que tenga un búfer lo suficientemente grande como para contener cualquier mensaje reensamblado. Por lo tanto, tiene que estar preparada para procesar los segmentos individualmente.

En los mensajes que están segmentados, esta aplicación no empieza a procesar un segmento hasta que todos los segmentos que constituyen el mensaje lógico están presentes. Por tanto, se especifica MQGMO_ALL_SEGMENTS_AVAILABLE para el primer segmento. Si se especifica MQGMO_LOGICAL_ORDER y hay un mensaje lógico actual, se ignora MQGMO_ALL_SEGMENTS_AVAILABLE.

Una vez recuperado el primer segmento de un mensaje lógico, use MQGMO_LOGICAL_ORDER para garantizar que los segmentos restantes del mensaje lógico se recuperen en orden.

No se tienen en cuenta los mensajes de grupos distintos. Si tienen lugar dichos mensajes, se procesan en el orden en el que se produce el primer segmento de cada mensaje en la cola.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentación de aplicación de un mensaje lógico*

Los mensajes tienen que mantenerse en un orden lógico dentro de un grupo y algunos, o todos ellos, pueden ser tan grandes que requieran una segmentación de aplicación.

En este ejemplo, se va a colocar un grupo de cuatro mensajes lógicos. Menos el tercero todos son grandes y requieren una segmentación, que la lleva a cabo la aplicación colocadora:

```
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
```

En la aplicación obtenedora, se especifica MQGMO_ALL_MSGS_AVAILABLE en el primer MQGET. Esto significa que no se recuperará ningún mensaje ni segmento de un grupo mientras no esté disponible todo el grupo. Cuando se recupera el primer mensaje físico de un grupo, se utiliza MQGMO_LOGICAL_ORDER para garantizar que los segmentos y los mensajes del grupo se recuperan en orden:

```
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
```

Nota: Si se especifica MQGMO_LOGICAL_ORDER y hay un grupo actual, se ignora MQGMO_ALL_MSGS_AVAILABLE.

Mensajes de referencia
Utilice esta información para obtener más información sobre los mensajes de referencia.

Nota: No está soportado en IBM MQ for z/OS.

Este método permite que se pueda transferir un objeto grande de un nodo a otro sin almacenar el objeto en colas IBM MQ ya sea en los nodos de origen o de destino. Esto es especialmente ventajoso cuando los datos existen en otro formato, por ejemplo, para aplicaciones de correo.

Para ello, especifique una salida de mensajes en ambos extremos de un canal. Para obtener información sobre cómo conseguirlo consulte el apartado "Programas de salida de mensajes de canal" en la página 1004.

IBM MQ define el formato de una cabecera de mensaje de referencia (MQRMH). Consulte MQRMH para ver una descripción. Este se reconoce con un nombre de formato definido y puede ir seguido por datos reales.

Para iniciar la transferencia de un objeto grande, una aplicación puede poner un mensaje que conste de una cabecera de mensaje de referencia pero no contenga datos tras ella. Cuando este mensaje deje el nodo, la salida de mensajes recuperará el objeto de la manera adecuada y lo añadirá al mensaje de referencia. A continuación, devolverá el mensaje (de mayor tamaño ahora que antes) al agente de canal de mensajes de envío para su transmisión al MCA de recepción.

Se configura otra salida de mensajes en el MCA de recepción. Cuando esta salida de mensajes reciba uno de estos mensajes, creará el objeto utilizando los datos de objeto que se hayan añadido y lo pasará al mensaje de referencia *sin* ellos. Ahora, las aplicaciones pueden recibir el mensaje de referencia y sabrán que el objeto (o al menos la parte representada por este mensaje de referencia) se ha creado en este nodo.

La cantidad máxima de datos de objeto que una salida de mensajes de envío puede añadir al mensaje de referencia está limitada por la longitud de mensaje máxima negociada para el canal. La salida solo puede devolver un mensaje individual al MCA por cada mensaje que se pase, por lo que la aplicación que lo pone puede poner varios mensajes para provocar que se transfiera un objeto. Cada mensaje debe identificar la longitud *lógica* y el desplazamiento del objeto que se le debe añadir. No obstante, en los casos donde no es posible saber el tamaño total del objeto o el tamaño máximo permitido por el canal, diseñe la salida de mensajes de envío de manera que la aplicación que pone mensajes solo ponga un mensaje individual, y la propia salida ponga el siguiente mensaje en la cola de transmisión cuando se hayan añadido tantos datos como sea posible al mensaje que se ha pasado.

Antes de utilizar este método para gestionar mensajes grandes, tenga en cuenta los puntos siguientes:

- El MCA y la salida de mensajes se ejecutan con un ID de usuario de IBM MQ. La salida de mensajes (y, por tanto, el ID de usuario) necesita acceder al objeto para recuperarlo en el extremo de envío o para crearlo en el extremo de recepción; esto podría ser factible únicamente en los casos en que el objeto sea universalmente accesible. Esto crea un problema de seguridad.
- Si el mensaje de referencia con datos en masa añadidos debe viajar a través de varios gestores de colas antes de llegar a su destino, los datos masivos se presentan en colas IBM MQ en los nodos que intervienen. No obstante, no es necesario que se proporcionen salidas ni soporte especial en estos casos.
- El diseño de su salida de mensajes se dificulta si se permiten redirecciones o colas de mensajes no entregados. En estos casos, las partes del objeto podrían llegar desordenadas.
- Cuando un mensaje de referencia llega a su destino, la salida de mensajes de recepción crea el objeto. No obstante, no está sincronizado con la unidad de trabajo del MCA, por lo que si el lote se restituye, otro mensaje de referencia que contiene esta misma parte del objeto llegará en un lote posterior y la salida de mensajes intentará volver a crear la misma parte del objeto. Si el objeto es, por ejemplo, una serie de actualizaciones de base de datos, esto podría ser inaceptable. Si es así, la salida de mensajes debe mantener un registro de las actualizaciones que se han aplicado; esto puede requerir el uso de una cola IBM MQ.

- En función de las características del tipo de objeto, es posible que sea necesario que las salidas de mensajes y las aplicaciones cooperen para mantener recuentos de uso, de manera que se pueda suprimir el objeto cuando ya no sea necesario. Es posible que también se necesite un identificador de instancia; se proporciona un campo para esto en la cabecera de mensaje de referencia (consulte MQRMH).
- Si se pone un mensaje de referencia como una lista de distribución, el objeto debe ser recuperable para cada lista de distribución resultante o destino individual de dicho nodo. Es posible que necesite mantener recuentos de uso. Tenga en cuenta también la posibilidad de que un nodo pueda ser el nodo final para algunos de los destinos de la lista, pero que sea un nodo intermedio para otros.
- Normalmente, los datos en masa no se convierten. Esto se debe a que la conversión se lleva a cabo *antes* de que se invoque la salida de mensajes. Por este motivo, no se debe solicitar la conversión en el canal emisor que los origina. Si el mensaje de referencia pasa a través de un nodo intermedio, los datos masivos se convierten cuando se envían desde el nodo intermedio, si se solicita.
- Los mensajes de referencia no se pueden segmentar.

Utilización de las estructuras MQRMH y MQMD

Consulte MQRMH y MQMD para ver una descripción de los campos de la cabecera de mensaje de referencia y el descriptor de mensaje.

En la estructura MQMD, establezca el campo *Format* en MQFMT_REF_MSG_HEADER. El formato de MQHREF, cuando se solicita en MQGET, lo convierte IBM MQ de forma automática junto con los datos masivos que le sigan.

A continuación se muestra un ejemplo del uso de los campos *DataLogicalOffset* y *DataLogicalLength* de MQRMH:

Una aplicación de transferencia podría poner un mensaje de referencia con:

- Ningún dato físico
- *DataLogicalLength* = 0 (este mensaje representa el objeto completo)
- *DataLogicalOffset* = 0.

Suponiendo que el objeto tiene una longitud de 70 000 bytes, la salida de mensajes de envío enviará los primeros 40 000 bytes a lo largo del canal en un mensaje de referencia que contiene:

- 40 000 bytes de datos físicos que siguen a MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (desde el comienzo del objeto).

A continuación, coloca otro mensaje en la cola de transmisión, que contiene:

- Ningún dato físico
- *DataLogicalLength* = 0 (al final del objeto). Puede especificar aquí un valor de 30 000.
- *DataLogicalOffset* = 40000 (a partir de este punto).

Cuando la salida de mensajes de envío haya visto esta salida de mensajes, se añadirán los 30 000 bytes de datos restantes y los campos se establecerán en:

- 30 000 bytes de datos físicos que siguen a MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (a partir de este punto).

También se establece el distintivo MQRMHF_LAST.

Para ver una descripción de los programas de ejemplo proporcionados para el uso de mensajes de referencia, consulte “Utilización de los programas de ejemplo en Multiplataformas” en la página 1086.

Espera de mensajes

Si desea que un programa espere hasta que llegue un mensaje a una cola, especifique la opción MQGMO_WAIT en el campo *Options* de la estructura MQGMO.

Utilice el campo *WaitInterval* de la estructura MQGMO para especificar tiempo máximo (en milisegundos) que quiere que la llamada MQGET espere la llegada de un mensaje a la cola.

Si el mensaje no llega en ese tiempo, la llamada MQGET se completa con el código de razón MQRC_NO_MSG_AVAILABLE.

Puede especificar un intervalo de espera ilimitado usando la constante MQWI_UNLIMITED en el campo *WaitInterval*. No obstante, puede haber sucesos que no controle y que podrían hacer que su programa espere mucho tiempo, por lo que debería utilizar esta constante con precaución. Las aplicaciones de IMS no debe especificar un intervalo de espera ilimitado porque se impediría la terminación del sistema IMS. (Cuando IMS termina, necesita que finalicen todas las regiones dependientes). En su lugar, las aplicaciones IMS pueden especificar un intervalo de espera finito; a continuación, si la llamada se completa sin recuperar un mensaje tras dicho intervalo, emita otra llamada MQGET con la opción de espera.

Nota: Si hay más de un programa esperando en la misma cola compartida para *eliminar* un mensaje, solo se activa un programa mediante la llegada de un mensaje. No obstante, si hay más de un programa a la espera para revisar un mensaje, se pueden activar todos los programas. Para obtener más información, consulte la descripción del campo *Options* de la estructura MQGMO en MQGMO.

Si el estado de la cola o del gestor de cola cambia antes de que caduque el intervalo de espera, se producen las acciones siguientes:

- Si el gestor de cola entra en estado de desactivación temporal, y ha utilizado la opción MQGMO_FAIL_IF_QUIESCING, la espera se cancela y la llamada MQGET se completa con el código de razón MQRC_Q_MGR_QUIESCING. Sin esta opción, la llamada permanece a la espera.
- z/OS En z/OS, si la conexión (para una aplicación CICS o IMS) entra en estado de desactivación temporal, y utiliza la opción MQGMO_FAIL_IF_QUIESCING, la espera se cancela y la llamada MQGET se completa con el código de razón MQRC_CONN_QUIESCING. Sin esta opción, la llamada permanece a la espera.
- Si el gestor de cola se ve forzado a parar, o se cancela, la llamada MQGET se completa con los códigos de razón MQRC_Q_MGR_STOPPING o MQRC_CONNECTION_BROKEN.
- Si los atributos en la cola (o una cola en la que se resuelve el nombre de cola) se cambia de forma que obtenga solicitudes se inhibe ahora, la espera se cancela y la llamada MQGET se completa con el código de razón MQRC_GET_INHIBITED.
- Si los atributos en la cola (o una cola en la que se resuelve el nombre de cola) se cambia de forma que hace falta la opción FORCE, la espera se cancela y la llamada MQGET se completa con el código de razón MQRC_OBJECT_CHANGED.

z/OS Si quiere que su aplicación espere en más de una cola, utilice el recurso de señal de IBM MQ for z/OS (consulte “Uso de señales” en la página 818). Para obtener más información sobre las circunstancias en las que se producen estas acciones, consulte MQGMO.

Uso de señales

La señalización solo está soportada en IBM MQ for z/OS.

El uso de señales es una opción en la llamada MQGET que permite al sistema operativo notificar (o *enviar una señal*) a un programa cuando un mensaje esperado llega a una cola. Esto es similar a la función *obtener con espera* descrita en el tema “Espera de mensajes” en la página 818, porque permite a un programa seguir con otro trabajo mientras espera la señal. No obstante, si se usan las señales, se puede liberar la hebra de aplicación en el sistema operativo para notificar al programa la llegada de un mensaje.

Configuración de una señal

Para configurar una señal, haga lo siguiente en la estructura MQGMO que use en la llamada MQGET:

1. Defina la opción MQGMO_SET_SIGNAL en el campo *Options*.
2. Defina la vida máxima de la señal en el campo *WaitInterval*. Esto establece la cantidad de tiempo (en milisegundos) durante la cual se desea que IBM MQ supervise la cola. Use el valor MQWI_UNLIMITED para especificar una vida ilimitada.

   Nota: Las aplicaciones IMS no pueden especificar un intervalo de tiempo ilimitado porque ello impediría que el sistema IMS terminase. (Cuando IMS termina, necesita que finalicen todas las regiones dependientes). En su lugar, las aplicaciones IMS pueden examinar el estado del ECB a intervalos regulares (véase el paso 3). Un programa puede tener señales configuradas en varios manejadores de cola a la vez:
3. Especifique la dirección del *bloque de control de suceso* (Event Control Block, ECB) en el campo *Signal1*. Esto le notificará el resultado de la señal. El almacenamiento ECB tiene que estar disponible mientras no se cierre la cola.

Nota: No se puede usar la opción MQGMO_SET_SIGNAL con la opción MQGMO_WAIT.

Cuando llega el mensaje

Cuando llega un mensaje adecuado, se devuelve un código de terminación al ECB.

El código de terminación describe uno de los eventos siguientes:

- El mensaje para el que se ha configurado la señal ha llegado a la cola. El mensaje no se reserva al programa que solicita una señal, de ahí que dicho programa tenga que volver a emitir una llamada MQGET para obtener el mensaje.

  Nota: Otra aplicación podría obtener el mensaje en el intervalo de tiempo que transcurre entre la recepción de la señal y la emisión de otra llamada MQGET.
- El intervalo de espera configurado se ha agotado y el mensaje para el que se ha configurado la señal no ha llegado a la cola. IBM MQ ha cancelado la señal.
- La señal se ha cancelado. Esto ocurre, por ejemplo, si el gestor de colas se para, o si cambia el atributo de la cola de forma que ya no se permitan más llamadas MQGET.

Cuando hay un mensaje adecuado listo en la cola, la llamada MQGET termina de la misma forma que una llamada MQGET sin señalización. Asimismo, si se detecta un error inmediatamente, la llamada termina y se establecen los códigos de retorno.

Cuando la llamada se acepta y no hay ningún mensaje disponible inmediatamente, se devuelve el control al programa para que pueda seguir con otro trabajo. Ninguno de los campos de salida del descriptor de mensaje está definido, pero el parámetro **CompCode** está estabecido a QCC_WARNING y el parámetro **Reason** etá establecido a MQRC_SIGNAL_REQUEST_ACCEPTED.

Para obtener información relativa a lo que IBM MQ puede devolver a una aplicación cuando hace una llamada MQGET usando señales, consulte MQGET.

Si el programa no tiene ningún otro trabajo que hacer mientras espera la publicación del ECB, puede esperar a dicho ECB usando:

- En un programa de CICS Transaction Server for z/OS, el comando EXEC CICS WAIT EXTERNAL.
- En programas por lotes y de IMS, la macro WAIT de z/OS.

Si cambia el estado de la cola o del gestor de colas mientras se configura la señal (es decir, el ECB aún no se ha publicado), tienen lugar las acciones siguientes:

- Si el gestor de colas entra en estado de desactivación temporal y se ha usado la opción MQGMO_FAIL_IF_QUIESCING, la señal se cancela. El ECB se publica con el código de terminación MQEC_Q_MGR_QUIESCING. Sin esta opción, la señal permanece configurada.
- Si se fuerza una parada del gestor de colas, o si este se cancela, la señal se cancela. La señal se entrega con el código de terminación MQEC_WAIT_CANCELED.

- Si los atributos de la cola (o una cola a la que resuelva el gestor de colas) se cambian de forma que las peticiones de obtención queden inhibidas, la señal se cancela. La señal se entrega con el código de terminación MQEC_WAIT_CANCELED.

Nota:

1. Si más de un programa ha configurado una señal en la misma cola compartida para eliminar un mensaje, solo se activa un programa mediante una llegada de mensaje. No obstante, si hay más de un programa a la espera para revisar un mensaje, se pueden activar todos los programas. Las reglas a las que se atiene el gestor de colas a la hora de decidir qué aplicaciones se activan son las mismas que las de las aplicaciones en espera: para obtener información adicional, consulte la descripción del campo *Options* de la estructura MQGMO en MQGMO - Opciones del mensaje de obtención.
2. Si hay más de una llamada MQGET a la espera del mismo mensaje, con una combinación de opciones de espera y señal, cada llamada en espera se considera igual. Para obtener información adicional, consulte la descripción del campo *Options* de la estructura MQGMO en MQGMO - Opcions del mensaje de obtención.
3. En determinadas circunstancias, es posible que una llamada MQGET recupere un mensaje y que una señal (resultante de la llegada del mismo mensaje) se entregue. Esto significa que, cuando el programa emita otra llamada MQGET (porque la señal se haya entregado), podría no haber mensajes disponibles. Diseñe el programa para probar esta situación.

Para obtener información sobre cómo configurar una señal, consulte la descripción de la opción MQGMO_SET_SIGNAL y el campo *Signal1* en Signal1.

Omisión de restitución

Puede impedir que un programa de aplicación entre en un bucle *MQGET-error-restitución* especificando la opción **MQGMO_MARK_SKIP_BACKOUT** en la llamada MQGET.

Nota: Solo está soportado en IBM MQ for z/OS.

Como parte de una unidad de trabajo, un programa de aplicación puede emitir una o varias llamadas MQGET para obtener mensajes de una cola. Si el programa de aplicación detecta un error, puede restituir la unidad de trabajo. Como resultado, se restauran todos los recursos actualizados durante esa unidad de trabajo al estado en el que estaban antes de que se iniciara la unidad de trabajo, y se restablecen los mensajes recuperados por las llamadas MQGET.

Una vez restablecidos, estos mensajes están disponibles para las siguientes llamadas MQGET emitidas por el programa de aplicación. En muchos casos, esto no supone ningún problema para el programa de aplicación. No obstante, en los casos en los que el error que provoca la restitución no puede eludirse, el restablecimiento del mensaje en la cola puede hacer que el programa de aplicación entre en un bucle *MQGET-error-restitución*.

Para evitar este problema, especifique la opción MQGMO_MARK_SKIP_BACKOUT en la llamada MQGET. Esto marca la solicitud MQGET como no implicada en una restitución iniciada por la aplicación y, por lo tanto, no debe restituirse. El uso de esta opción implica que cuando se produce una restitución, las actualizaciones de los demás recursos se restituyen según sea necesario, pero el mensaje marcado se trata como si se hubiera recuperado en una nueva unidad de trabajo.

El programa de aplicación debe emitir una llamada de IBM MQ para confirmar la nueva unidad de trabajo o restituir la nueva unidad de trabajo. Supongamos que el programa ejecuta el manejo de excepciones, por ejemplo, informa al originador de que el mensaje se ha descartado y confirma la unidad de trabajo, esto elimina el mensaje de la cola. Si la nueva unidad de trabajo se restituye (por algún motivo), el mensaje se restablece en la cola.

En una unidad de trabajo, solo puede haber una solicitud MQGET marcada con la omisión de restitución; no obstante, puede haber otros mensajes que no estén marcados con la omisión de restitución. Si un mensaje se marca con la omisión de restitución, las próximas llamadas MQGET en la unidad de trabajo que especifiquen MQGMO_MARK_SKIP_BACKOUT fallarán con el código de razón MQRC_SECOND_MARK_NOT_ALLOWED.

Nota:

1. El mensaje marcado solo omite la restitución si una solicitud de aplicación termina la unidad de trabajo que lo contiene para restituirlo. Si la unidad de trabajo se restituye por cualquier otro motivo, el mensaje se restituye en la cola de la misma forma que lo haría si no se hubiera marcado para omitir la restitución.
2. La omisión de restitución no está soportada en los procedimientos almacenados de Db2 que participan en unidades de trabajo controladas por RRS. Por ejemplo, una llamada MQGET con la opción MQGMO_MARK_SKIP_BACKOUT fallará con el código de razón MQRC_OPTION_ENVIRONMENT_ERROR.

La Figura 63 en la página 821 ilustra una secuencia típica de pasos que puede contener un programa de aplicación cuando una solicitud MQGET debe omitir la restitución.

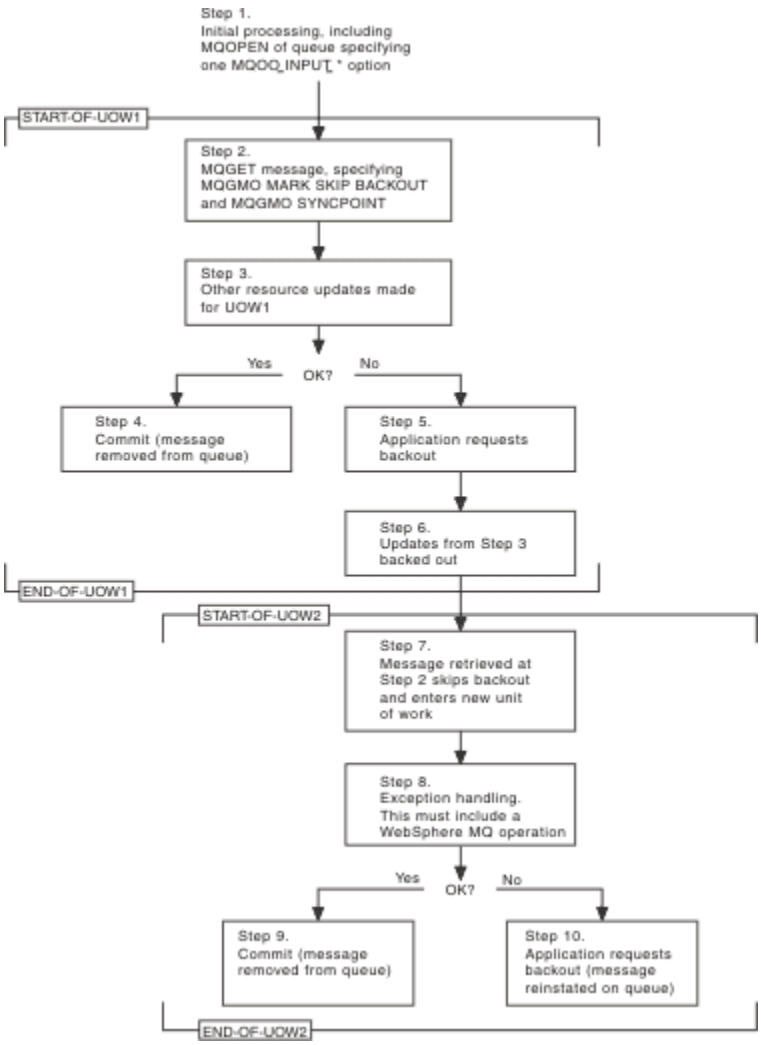


Figura 63. Omisión de restitución utilizando MQGMO_MARK_SKIP_BACKOUT

Los pasos de la Figura 63 en la página 821 son:

Paso 1
El proceso inicial se produce en la transacción, incluida una llamada MQOPEN para abrir la cola (especificando una de las opciones MQOO_INPUT_* para poder obtener los mensajes de la cola en el paso 2).

Paso 2
Se llama a MQGET con MQGMO_SYNCPOINT y MQGMO_MARK_SKIP_BACKOUT. MQGMO_SYNCPOINT es necesario porque MQGET debe estar dentro de una unidad de trabajo para que MQGMO_MARK_SKIP_BACKOUT sea eficaz. En la Figura 63 en la página 821, esta unidad de trabajo se conoce como UOW1.

Paso 3
Se realizan otras actualizaciones de recursos como parte de UOW1. Estas pueden incluir más llamadas MQGET (emitidas sin MQGMO_MARK_SKIP_BACKOUT).

Paso 4
Todas las actualizaciones de los pasos 2 y 3 finalizan según es necesario. El programa de aplicación confirma las actualizaciones y UOW1 finaliza. El mensaje recuperado en el paso 2 se elimina de la cola.

Paso 5
Algunas de las actualizaciones de los pasos 2 y 3 no finalizan según es necesario. El programa de aplicación solicita que las actualizaciones realizadas durante estos pasos se restituyan.

Paso 6
Las actualizaciones realizadas en el paso 3 se restituyen.

Paso 7
La solicitud MQGET realizada en el paso 2 omite la restitución y pasa a formar parte de una nueva unidad de trabajo, UOW2.

Paso 8
UOW2 ejecuta el manejo de excepciones como respuesta a la restitución de UOW1. (Por ejemplo, una llamada MQPUT a otra cola, que indica que se ha producido un problema que ha hecho que se restituya OUW1).

Paso 9
El paso 8 finaliza según es necesario, el programa de aplicación confirma la actividad y UOW2 finaliza. Como la solicitud MQGET forma parte de UOW2 (véase el paso 7), debido a esta confirmación, el mensaje se elimina de la cola.

Paso 10
El paso 8 no finaliza según es necesario y el programa de aplicación restituye UOW2. Como la solicitud de obtención de mensaje forma parte de UOW2 (véase el paso 7), también se restituye y se restablece en la cola. Ahora está disponible para otras llamadas MQGET emitidas por este programa de aplicación u otros (de la misma forma que cualquier mensaje de la cola).

Conversión de datos de aplicación

Cuando es necesario, los MCA convierten el descriptor de mensaje y los datos de cabecera al juego de caracteres y codificación pertinentes. Ambos extremos del enlace (es decir, el MCA local o el MCA remoto) pueden realizar la conversión.

Cuando una aplicación transfiere mensajes a una cola, el gestor de colas local añade información de control a los descriptores de mensaje para facilitar el control de los mensajes cuando los gestores de colas y los MCA los procesan. En función del entorno, los campos de datos de cabecera de mensaje se crean en el juego de caracteres y en la codificación del sistema local.

Al mover mensajes entre sistemas, a veces es necesario convertir los datos de aplicación al juego de caracteres y la codificación que requiere el sistema receptor. Esto se puede hacer desde los programas de aplicación en el sistema receptor, o a través de los MCA en el sistema emisor. Si el sistema receptor da soporte a la conversión de datos, utilice los programas de aplicación para convertir los datos de aplicación, en lugar de depender de la conversión que ya se haya producido en el sistema emisor.

Los datos se aplicación se convierten en un programa de aplicación cuando especifica la opción MQGMO_CONVERT en el campo *Options* de la estructura MQGMO que se ha pasado a una llamada MQGET y cuando *todas* las sentencias siguientes son verdaderas:

- Los campos *CodedCharSetId* o *Encoding* establecidos en la estructura MQMD asociada al mensaje de la cola difieren de los campos *CodedCharSetId* o *Encoding* establecidos en la estructura MQMD especificada en la llamada MQGET.
- El campo *Format* de la estructura MQMD asociada al mensaje no es MQFMT_NONE.
- El parámetro *BufferLength* especificado en la llamada MQGET no es cero.
- La longitud de los datos del mensaje no es cero.

- El gestor de colas da soporte a la conversión entre los campos *CodedCharSetId* y *Encoding* especificados en la estructura MQMD asociada al mensaje y la llamada MQGET. Consulte CodedCharSetId y Codificación para conocer detalles sobre los identificadores de juegos de caracteres codificados y las codificaciones de máquina soportados.
- El gestor de colas da soporte a la conversión del formato de mensaje. Si el campo *Format* de la estructura MQMD asociada al mensaje es uno de los formatos incorporados, el gestor de colas puede convertir el mensaje. Si el valor de *Format* no es uno de los formatos incorporados, debe escribir una salida de conversión de datos para poder convertir el mensaje.

Si es el MCA emisor el encargado de convertir los datos, especifique la palabra clave CONVERT(YES) en la definición de cada canal emisor o de servidor para el que se necesite la conversión. Si la conversión de datos falla, el mensaje se envía a la DLQ del gestor de colas emisor, y en el campo *Feedback* de la estructura MQDLH se indica el motivo. Si no se puede colocar el mensaje en la DLQ el canal se cierra, y el mensaje no convertido permanece en la cola de transmisión. La conversión de datos dentro de las aplicaciones en lugar durante el envío de los MCA, evita esta situación.

Como regla, los datos de mensaje que el formato incorporado o la salida de conversión de datos describen como datos de *carácter*, se convertirán del juego de caracteres codificado que utiliza el mensaje al que se haya solicitado, y los campos *numéricos* se convierte a la codificación solicitada.

Para obtener más detalles de los convenios de proceso de conversión utilizados al convertir los formatos incorporados, y para obtener información sobre cómo escribir sus propias salidas de conversión de datos, consulte “Escribir salidas de conversión de datos” en la página 1009. Consulte también Idiomas nacionales y Codificaciones de máquina para obtener información sobre las tablas de soporte de idiomas y las codificaciones de máquina soportadas.

Conversión de caracteres de nueva línea EBCDIC

Si tiene que asegurarse de que los datos que se envían desde una plataforma EBCDIC a una ASCII sean idénticos a los datos que se devuelven, debe controlar la conversión de los caracteres de nueva línea EBCDIC.

Puede hacerlo utilizando un conmutador dependiente de la plataforma que fuerza a IBM MQ a que utilice tablas de conversión no modificadas, pero debe tener en cuenta que el comportamiento puede ser incoherente.

El problema surge porque el carácter de nueva línea EBCDIC no se convierte de forma coherente entre las diferentes plataformas o tablas de conversión. Como resultado, si los datos se muestran en una plataforma ASCII, el formato puede ser incorrecto. Esto dificultaría, por ejemplo, el poder administrar un sistema IBM i de forma remota, desde una plataforma ASCII mediante RUNMQSC.

Consulte la sección Conversión de datos para obtener más información acerca de cómo convertir los datos con formato EBCDIC al formato ASCII.

Cómo examinar mensajes en una cola

Utilice esta información para aprender a examinar mensajes en una cola utilizando la llamada MQGET.

Para utilizar la llamada MQGET para examinar los mensajes en una cola:

1. Llame a MQOPEN para abrir la cola para examinar los mensajes, especificando la opción MQOO_BROWSE.
2. Para examinar el primer mensaje de la cola, llame a MQGET con la opción MQGMO_BROWSE_FIRST. Para encontrar el mensaje que desea, llame a MQGET repetidamente con la opción MQGMO_BROWSE_NEXT para recorrer varios mensajes.

   Debe establecer los campos *MsgId* y *CorrelId* de la estructura MQMD en nulo después de cada llamada MQGET para poder ver todos los mensajes.
3. Llame a MQCLOSE para cerrar la cola.

El cursor para examinar
Cuando se abre (MQOPEN) una cola para su examen, la llamada establece un cursor para examinar para la llamadas MQGET que usen una de las opciones de examen. Se puede pensar en el cursor para examinar como un un puntero lógico que se coloca delante del primer nombre de la cola.

Se puede tener más de un cursor para examinar activo (desde un único programa) emitiendo varias solicitudes MQOPEN a la misma cola.

Cuando se invoca MQGET para su examen, use una de las opciones siguientes en la estructura MQGMO:

MQGMO_BROWSE_FIRST
Obtiene una copia del primer mensaje que cumple las condiciones especificadas en la estructura MQMD.

MQGMO_BROWSE_NEXT
Obtiene una copia del siguiente mensaje que cumple las condiciones especificadas en la estructura MQMD.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Obtiene una copia del mensaje al que apunta en ese momento el cursor, es decir, el último que se recuperó con las opciones MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT.

En todos los casos, el mensaje permanece en la cola.

Cuando se abre una cola, el cursor para examinar está posicionado lógicamente justo antes del primer mensaje de la cola. Esto significa que, si se hace la llamada MQGET inmediatamente después de la llamada MQOPEN, se puede usar la opción MQGMO_BROWSE_NEXT para examinar el primer mensaje; no es necesario usar la opción MQGMO_BROWSE_FIRST.

El orden en que se copian los mensajes de la coa está determinado por el atributo **MsgDeliverySequence** de la cola. (Para obtener más información, consulte "Orden en el que se recuperan los mensajes de una cola" en la página 793).

- "Colas FIFO (primero en entrar, primero en salir)" en la página 824
- "Colas en secuencia de prioridad" en la página 824
- "Mensajes sin confirmar" en la página 825
- "Cambio de la secuencia de una cola" en la página 825
- "Uso de un índice de cola" en la página 825

Colas FIFO (primero en entrar, primero en salir)

El primer mensaje de una cola en esta secuencia es el mensaje que más tiempo ha estado en ella.

Use MQGMO_BROWSE_NEXT para leer los mensajes de forma secuencial en esta cola. Verá todos los mensajes colocados en la cola mientras esté examinando, ya que los mensajes en una cola con esta secuencia se colocan al final. Cuando el cursor llega al final de la cola, se queda donde está y retorna MQRC_NO_MSG_AVAILABLE. Se puede dejar ahí a la espera de más mensajes o restablecerlo al principio de la cola con la llamada MQGMO_BROWSE_FIRST.

Colas en secuencia de prioridad

El primer mensaje de una cola en esta secuencia es el mensaje que más tiempo ha estado en ella, el más largo y el de mayor prioridad en el momento de invocarse la llamada MQOPEN.

Use MQGMO_BROWSE_NEXT para leer los mensajes de la cola.

El cursor para examinar apunta al mensaje siguiente, trabajando desde la prioridad del primer mensaje para terminar con el mensaje de prioridad más baja. Examina todos los mensajes colocados durante este tiempo siempre que tengan una prioridad igual a, o menor que, la del mensaje identificado por el cursor para examinar actual.

Cualquier mensaje colocado en la cola que tenga una prioridad superior solo podrá examinarse de estas formas:

- Volviendo a abrir la cola para su examen, momento en el cual se establece un nuevo cursor para examinar.
- Usando la opción MQGMO_BROWSE_FIRST.

Mensajes sin confirmar

Un mensaje sin confirmar nunca será visible en un examen; el cursor para examinar lo pasa por alto.

Los mensajes que estén dentro de una unidad de trabajo no se podrán examinar mientras esta no se confirme. Los mensajes no cambian su posición en la cola cuando se confirman, de forma que los mensajes omitidos no confirmados no se verán, incluso si *son* confirmados, a menos que se vuelva a usar la opción MQGMO_BROWSE_FIRST.

Cambio de la secuencia de una cola

Si se cambia la secuencia de una cola de prioridad a FIFO mientras quedan mensajes en ella, el orden de los mensajes que ya están encolados no cambia. Los mensajes añadidos a la cola posteriormente recibirán la prioridad predeterminada de la cola.

Uso de un índice de cola

Cuando se examina una cola indexada que solo contiene mensajes de una única prioridad (ya sean persistente, no persistentes, o ambas cosas a la vez) el gestor de colas usa el índice para examinar cuando se usan determinadas formas de examen.

Nota: Solo está soportado en IBM MQ for z/OS.

Se usa cualquiera de las formas siguientes de examen cuando una cola indexada solo contiene mensajes de una única prioridad:

1. Si la cola está indexada por MSGID, las peticiones de examen que pasen un MSGID en la estructura MQMD se procesarán usando el índice para encontrar el mensaje de destino.
2. Si la cola está indexada por CORRELID, las peticiones de examen que pasen un CORRELID en la estructura MQMD se procesarán usando el índice para encontrar el mensaje de destino.
3. Si la cola está indexada por GROUPID, las peticiones de examen que pasen un GROUPID en la estructura MQMD se procesarán usando el índice para encontrar el mensaje de destino.

Si la petición de examen no pasa un MSGID, CORRELID ni GROUPID en la estructura MQMD, la cola se indexa y se devuelve un mensaje, se tiene que encontrar la entrada de índice del mensaje y usarse la información contenida en ella para actualizar el el cursor para examinar. Si se usa una amplia selección de valores de índice, esto no supone una cantidad adicional de procesamiento significativa en la petición de examen.

Explorar mensajes cuando se desconoce la longitud del mensaje

Para explorar un mensaje cuando se desconoce el tamaño del mensaje y no desea utilizar los campos *MsgId*, *CorrelId* o *GroupId* para localizar el mensaje, puede utilizar la opción MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Emita una MQGET con:
   - La opción MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT
   - La opción MQGMO_ACCEPT_TRUNCATED_MSG
   - Una longitud de almacenamiento intermedio de cero

   Nota: Si es probable que otro programa obtenga el mismo mensaje, puede considerar la posibilidad de utilizar también la opción MQGMO_LOCK. Se deberá devolver MQRC_TRUNCATED_MSG_ACCEPTED.
2. Utilice el valor de *DataLength* devuelto para asignar el almacenamiento necesario.
3. Emita una MQGET con MQGMO_BROWSE_MSG_UNDER_CURSOR.

El mensaje al que se apunto es el último que se ha recuperado. El cursor de exploración no se habrá movido. Puede optar por bloquear el mensaje con la opción MQGMO_LOCK o desbloquear un mensaje bloqueado con la opción MQGMO_UNLOCK.

La llamada falla si se ha emitido correctamente ninguna MQGET con las opciones MQGMO_BROWSE_FIRST o MQGMO_BROWSE_NEXT desde que se ha abierto la cola.

Eliminación de un mensaje que ha examinado
Puede eliminar de la cola un mensaje que ya ha examinado, siempre que haya abierto la cola para eliminar mensajes además de para examinarlos. (Debe especificar una de las opciones MQOO_INPUT_*, así como la opción MQOO_BROWSE, en la llamada MQOPEN).

Para eliminar el mensaje, llame a MQGET de nuevo, pero en el campo *Options* de la estructura MQGMO, especifique MQGMO_MSG_UNDER_CURSOR.En este caso, la llamada MQGET ignora los campos *MsgId*, *CorrelId* y *GroupId* de la estructura MQMD.

En el tiempo entre los pasos de examinar y eliminar, otro programa puede haber eliminado mensajes de la cola, incluido el mensaje en el cursor para examinar. En este caso, la llamada MQGET devuelve un código de razón para indicar que el mensaje no está disponible.

Examen de mensajes en orden lógico

"Ordenación lógica y física" en la página 794 explica la diferencia entre los órdenes lógico y físico de los mensajes de una cola. Esta distinción es de especial importancia cuando se examina una cola, porque, en general, los mensajes no se borran y las operaciones de examen no comienzan necesariamente por el principio de la cola.

Si una aplicación examina los diversos mensajes de un grupo (empleando un orden lógico), es importante seguir un orden lógico para alcanzar el comienzo del grupo siguiente, porque el último mensaje de un grupo podría encontrarse físicamente *después* del primer mensaje del grupo siguiente. La opción MQGMO_LOGICAL_ORDER garantiza que se siga un orden lógico al explorar una cola.

Use MQGMO_ALL_MSGS_AVAILABLE (o MQGMO_ALL_SEGMENTS_AVAILABLE) con cuidado en las operaciones de examen. Considere el caso de mensajes lógicos con MQGMO_ALL_MSGS_AVAILABLE. El efecto que esto tendría es que un mensaje lógico solo estaría disponible si todos los demás mensajes del grupo también estuvieran presentes. Si no estuvieran presentes, se saltaría el mensaje. Esto puede significar que, cuando luego lleguen los mensajes que faltan, pasarán inadvertidos a la siguiente operación browse-next (examinar siguiente).

Por ejemplo, si los siguientes mensajes lógicos están presentes:

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

y se emite una función de examen con MQGMO_ALL_MSGS_AVAILABLE, se devuelve el primer mensaje lógico 456, dejando el cursor para examinar en este mensaje lógico. Si ahora llega el segundo (último) mensaje del grupo 123:

```
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
```

y se emite la misma función browse-next, pasa desapercibido que el grupo 123 está ahora completo, porque el primer mensaje de este grupo está noticed that group 123 is now complete, because the first message *antes* del cursor para examinar.

En algunos casos (por ejemplo, si los mensajes se recuperan de forma destructiva cuando el grupo está presente en su totalidad), también se puede usar MQGMO_ALL_MSGS_AVAILABLE junto con MQGMO_BROWSE_FIRST. En caso contrario, hay que repetir la exploración de examen para tomar nota de los mensajes recién llegados que se han pasado por alto; no basta con emitir MQGMO_BROWSE_NEXT

y MQGMO_ALL_MSGS_AVAILABLE para tenerlos en cuenta. (Esto también sucede a los mensajes de prioridad más alta que lleguen una vez finalizada la exploración de los mensajes).

Las secciones siguientes se tratan ejemplos de examen que manejan mensajes sin segmentar; los mensajes segmentados se atienen a principios similares.

Revisión de mensajes en grupos
En este ejemplo, la aplicación revisa cada mensaje de la cola, en orden lógico.

Los mensajes en la cola pueden estar agrupados. Para los mensajes agrupados, la aplicación no empieza a procesar ningún grupo hasta que hayan llegado todos los mensajes. Por lo tanto, se especifica MQGMO_ALL_MSGS_AVAILABLE para el primer mensaje del grupo; para los mensajes posteriores del grupo, esta opción no es necesaria.

MQGMO_WAIT se utiliza en este ejemplo. No obstante, aunque la espera se puede atender si llegan grupos nuevos, por los motivos de "Examen de mensajes en orden lógico" en la página 826, no se atiende si el cursor de revisión ya ha pasado el primer mensaje lógico de un grupo, y entonces llegan los mensajes restantes. En cualquier caso, la espera de un intervalo apropiado asegura que la aplicación no entra en bucle constante mientras espera mensajes o segmentos nuevos.

MQGMO_LOGICAL_ORDER se utiliza en todo el proceso, para asegurarse de que la exploración se realiza en orden lógico. Esto contrasta con el ejemplo MQGET destructivo, en el que cada grupo se elimina, MQGMO_LOGICAL_ORDER no se utiliza cuando se busca el primer (o único) mensaje del grupo.

Se presupone que el almacenamiento intermedio de la aplicación siempre es suficientemente grande para almacenar el mensaje completo, tanto si el mensaje se ha segmentado como si no. Por tanto, MQGMO_COMPLETE_MSG se especifica en cada MQGET.

En el ejemplo siguiente se proporciona la exploración de los mensajes lógicos en un grupo:

```
/* 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 */
   ...
```

El grupo se repite hasta que se devuelve MQRC_NO_MSG_AVAILABLE.

Explorar y recuperar de forma destructiva
En este ejemplo, la aplicación explora cada uno de los mensajes lógicos de un grupo, antes de decidir si recupera este grupo de forma destructiva.

La primera parte de este ejemplo es similar al anterior. No obstante, en este caso, después de explorar un grupo completo, se decide regresar y recuperarlo de forma destructiva.

Dado que cada grupo se elimina en este ejemplo, no se utiliza MQGMO_LOGICAL_ORDER cuando se busca el primer o único mensaje de un grupo.

El siguiente es un ejemplo de explorar y, a continuación, recuperar de forma destructiva:

```
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 */
...
```

Cómo evitar una entrega reiterada de mensajes examinados
Mediante determinadas opciones de apertura y de obtención de mensajes, estos pueden marcarse como examinados para que las aplicaciones colaborativas actuales u otras no vuelvan a recuperarlos. Los mensajes pueden desmarcarse de forma explícita o automática para que se puedan volver a examinar.

Si se examinan los mensajes de una cola, puede que se recuperen en un orden distinto del orden en que se recuperarían si se obtuvieran de forma destructiva. En concreto, un mismo mensaje se puede examinar múltiples veces, lo que no es posible cuando se elimina de la cola. Para evitar esto, un mensaje se puede *marcar* cuando se examina, e impedirse la recuperación de los mensajes marcados. Esto se conoce a veces como *examen con marca*. Para marcar mensajes examinados, use la opción de obtención de mensaje MQGMO_MARK_BROWSE_HANDLE y para recuperar únicamente los mensajes no marcados, use MQGMO_UNMARKED_BROWSE_MSG. Si se usa la combinación de opciones MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG y MQGMO_MARK_BROWSE_HANDLE, y se emiten de forma repetida varios MQGET, se recuperará cada mensaje de la cola por turno. Esto impide una entrega repetida de mensajes aunque se use MQGMO_BROWSE_FIRST para garantizar que no se salten mensajes. Esta combinación de opciones se puede representar mediante la única constante MQGMO_BROWSE_HANDLE. Cuando no quede ningún mensaje en la cola por examinar, se devolverá MQRC_NO_MSG_AVAILABLE.

Si hay varias aplicaciones examinando la misma cola, podrán abrir esta con las opciones MQOO_CO_OP y MQOO_BROWSE. El descriptor de objeto devuelto por cada MQOPEN se considera parte de un grupo cooperativo. Cualquier mensaje devuelto por una llamada MQGET que especifique la opción MQGMO_MARK_BROWSE_CO_OP se considerará marcado en este conjunto colaborativo de descriptores.

Si un mensaje ha estado marcado durante un tiempo, el gestor de colas puede desmarcarlo de forma automática para que vuelva a estar disponible en los exámenes. El atributo de gestor de colas MsgMarkBrowseInterval indica el tiempo en milisegundos durante el cual un mensaje permanece marcado para el conjunto colaborativo de descriptores. Un MsgMarkBrowseInterval de -1 significa que los mensajes nunca se desmarcan automáticamente.

Cuando el único proceso o el conjunto de procesos colaborativos dejen de marcar mensajes, los mensajes que estén marcados pasarán a estar desmarcados.

Ejemplos de examen colaborativo

Se podrían ejecutar múltiples copias de una aplicación distribuidora para examinar mensajes en una cola e iniciar un consumidor en función del contenido de cada mensaje. En cada distribuidora, abra la cola con with MQOO_CO_OP. Esto indica que las distribuidoras están cooperando y tendrán conocimiento de los mensajes marcados de cada una. Luego, cada distribuidora efectúa repetidas llamadas

MQGET especificando las opciones MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG y MQGMO_MARK_BROWSE_CO_OP (se puede usar la constante única MQGMO_BROWSE_CO_OP para representar esta combinación de opciones). Cada aplicación distribuidora recupera a continuación solo aquellos mensajes que aún no hayan sido marcados por otras distribuidoras colaborativas. La distribuidora inicializa un consumidor y le pasa el MsgToken devuelto por MQGET para que obtenga destructivamente el mensaje de la cola. Si el consumidor restituye el MQGET del mensaje, este estará disponible para que una de las examinadoras lo vuelva a distribuir, porque ya no está marcado. Si el consumidor no hace un MQGET del mensaje, una vez transcurrido el MsgMarkBrowseInterval, el gestor de colas lo desmarcará para el conjunto colaborativo de descriptores y se podrá volver a distribuir.

En lugar de múltiples copias de la misma distribuidora, se podrían tener una serie de aplicaciones distribuidoras distintas examinando la cola, siendo cada una de ellas adecuada para procesar un subconjunto de los mensajes de la cola. En cada distribuidora, abra la cola con with MQOO_CO_OP. Esto indica que las distribuidoras están cooperando y tendrán conocimiento de los mensajes marcados de cada una.

- Si el orden del procesamiento de mensajes en una determinada distribuidora es importante, cada distribuidora efectúa repetidas llamadas MQGET especificando las opciones MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG y MQGMO_MARK_BROWSE_HANDLE (o MQGMO_BROWSE_HANDLE). Si los mensajes examinados son adecuados para esta distribuidora, esta efectúa una llamada MQGET especificando MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP y el MsgToken devuelto por la llamada MQGET anterior. Si la llamada es satisfactoria, la distribuidora inicializa el consumidor y le pasa el MsgToken.
- Si el orden en el procesamiento de los mensajes no es importante y es de esperar que la distribuidora procese la mayoría de los mensajes que se encuentre, use las opciones MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG y MQGMO_MARK_BROWSE_CO_OP (o MQGMO_BROWSE_CO_OP). Si la distribuidora examinara un mensaje que no pudiera procesar, lo desmarcaría invocando MQGET con las opciones MQMO_MATCH_MSG_TOKEN y MQGMO_UNMARK_BROWSE_CO_OP, y con el MsgToken devuelto anteriormente.

Algunos casos en los que falla la llamada MQGET

Si se modifican determinados atributos de una cola utilizando la opción FORCE en un mandato entre el proceso de emisión de una llamada MQOPEN y una llamada MQGET, la llamada MQGET falla y devuelve el código de razón MQRC_OBJECT_CHANGED.

El gestor de colas marca el descriptor de objeto como no válido. Esto también sucede si los cambios se aplican a cualquier cola en cuyo nombre de cola se resuelve. Los atributos que afectan al descriptor de este modo se listan en la descripción de la llamada MQOPEN contenida en la sección MQOPEN. Si su llamada devuelve el código de razónMQRC_OBJECT_CHANGED, cierre la cola, vuelva a abrirla y, a continuación, vuelva a intentar obtener el mensaje.

Si están inhibidas las operaciones de obtención para una cola desde la que está intentando obtener mensajes, o cualquier cola en cuyo nombre de cola se resuelva, la llamada MQGET falla y devuelve el código de razónMQRC_GET_INHIBITED. Esto sucede incluso si utiliza la llamada MQGET para la exploración. Es posible que pueda obtener correctamente un mensaje si intenta la llamada MQGET más tarde, si la aplicación se ha diseñado de modo que otros programas cambien los atributos de las colas con regularidad.

Si se ha suprimido una cola dinámica, ya sea temporal o permanente, las llamadas MQGET que utilizan un descriptor de objetos adquirido previamente fallarán con el código de razón MQRC_Q_DELETED.

Escritura de aplicaciones de publicación/suscripción

Empezar a escribir aplicaciones de publicación/suscripción de IBM MQ.

Para obtener una visión general de los conceptos de publicación/suscripción, consulte Mensajería de publicación/suscripción.

Consulte los siguientes temas para obtener información sobre la escritura de distintos tipos de aplicaciones de publicación/suscripción:

- “Escribir aplicaciones de publicación” en la página 830
- “Escritura de aplicaciones de suscriptor” en la página 837
- “Ciclos de vida de publicación/suscripción” en la página 854
- “Propiedades de los mensajes de publicación/suscripción” en la página 859
- “Orden de los mensajes” en la página 861
- “Interceptación de publicaciones” en la página 861
- “Opciones de publicación” en la página 869
- “Opciones de suscripción” en la página 869

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

“Desarrollo de aplicaciones para IBM MQ” en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

“Consideraciones acerca del diseño de las aplicaciones IBM MQ” en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

“Desarrollo de una aplicación procedimental para encolamientos” en la página 740
Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

“Desarrollo de aplicaciones procedimentales cliente” en la página 936
Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

“Creación de una aplicación procedimental” en la página 1025
Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

“Tratamiento de errores en un programa procedimental” en la página 1064
Esta información explica los errores asociados a las llamadas MQI de una aplicación cuando se efectúa una llamada o cuando el mensaje se entrega en su destino final.

Tareas relacionadas

“Utilización de programas procedimentales de ejemplo de IBM MQ” en la página 1085
Estos programas de ejemplo se escriben en lenguajes de procedimiento y muestran los usos típicos de la interfaz de cola de mensajes (MQI). Programas IBM MQ en distintas plataformas.

Escribir aplicaciones de publicación

Comience a escribir aplicaciones de publicación estudiando dos ejemplos. El primero se ha diseñado para que sea lo más aproximado posible a una aplicación de punto a punto que transfiere mensajes a una cola y el segundo muestra cómo crear temas de forma dinámica, un patrón más común en las aplicaciones de publicación.

Escribir una aplicación de publicación simple de IBM MQ es como escribir una aplicación de punto a punto de IBM MQ que coloca mensajes en una cola (Tabla 121 en la página 831). La diferencia es que envía mensajes MQPUT a un tema, no a una cola.

Tabla 121. Comparación entre el patrón de programas de IBM MQ de punto a punto y el de publicación/ suscripción.

Paso	Llamada MQ punto a punto	Llamada MQ de publicación
Conectarse a un gestor de colas	MQCONN	MQCONN
Abrir cola	MQOPEN	
Abre tema		MQOPEN
Transferir mensaje(s)	MQPUT	MQPUT
Cerrar tema		MQCLOSE
Cerrar cola	MQCLOSE	
Desconectarse del gestor de colas	MQDISC	MQDISC

Para concretarlo, hay dos ejemplos de aplicaciones para publicar valores en bolsa. En el primer ejemplo ("Ejemplo 1: Publicador en un tema fijo" en la página 831), que se ha diseñado de forma muy aproximada a la transferencia de mensajes a una cola, el administrador crea una de definición de tema de modo similar al de la creación de una cola. El programador codifica MQPUT para que escriba los mensajes en el tema, en lugar de escribirlos en una cola. En el segundo ejemplo ("Ejemplo 2: aplicación de publicación en un tema variable" en la página 834), el patrón de interacción del programa con IBM MQ es similar. La diferencia es que es el programador quien proporciona el tema en el que se escribe el mensaje y no el administrador. En la práctica, normalmente esto significa que la serie de tema es contenido definido o proporcionado por otro origen, tal como una entrada realizada por una persona con un navegador.

Conceptos relacionados

"Escritura de aplicaciones de suscriptor" en la página 837
Iníciese en la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación de IBM MQ que consume mensaje de una cola, una aplicación que crea una suscripción y no requiere conocimiento sobre colas y, finalmente, un ejemplo que utiliza tanto colas como suscripciones.

Referencia relacionada

DEFINE TOPIC
DISPLAYTOPIC
DISPLAYTPSTATUS

Ejemplo 1: Publicador en un tema fijo
Un programa de IBM MQ que ilustra la publicación en un tema definido administrativamente.

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

Consulte la salida en la Figura 65 en la página 832

```
#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);
}
```

Figura 64. Publicador de IBM MQ simple en un tema fijo.

```
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
```

Figura 65. Salida de ejemplo del primer ejemplo de publicador

Las siguientes líneas de código ilustran aspectos de la escritura de una aplicación de publicador para IBM MQ.

char topicNameDefault[] = "IBMSTOCKPRICE";
Se define un nombre de tema predeterminado en el programa. Puede alterarlo temporalmente proporcionando el nombre de otro objeto de tema como primer argumento al programa.

MQCHAR resTopicStr[151];
`resTopicStr` apunta a `td.ResObjectString.VSPtr` y `MQOPEN` lo utiliza para devolver la serie de tema resuelta. Aumente en uno la longitud de `resTopicStr` para que sea más grande que la longitud pasada en `td.ResObjectString.VSBufSize` para dejar espacio para la terminación nula.

memset (resTopicStr, 0, sizeof(resTopicStr));
Inicialice `resTopicStr` en valores nulos para asegurarse de que la serie de tema resuelta que se devuelve en `MQCHARV` termine en nulos.

td.ObjectType = MQOT_TOPIC
Hay un nuevo tipo de objeto para la publicación/suscripción: el *objeto de tema*.

td.Version = MQOD_VERSION_4;
Para utilizar el nuevo tipo de objeto, debe utilizar al menos la *versión 4* del descriptor de objetos.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
`topicName` es el nombre de un objeto de tema, que a veces se denomina un objeto de tema administrativo. En el ejemplo, el objeto de tema debe crearse previamente utilizando IBM MQ Explorer o este mandato MQSC.

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
La serie de tema resuelta se repite en el `printf` final en el programa. Configure la estructura `MQCHARV ResObjectString` para IBM MQ para devolver la serie resuelta al programa.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
Abra el tema para la salida, al igual que se abre una cola para la salida.

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
Desea que los nuevos suscriptores puedan recibir la publicación y, al especificar `MQPMO_RETAIN` en el publicador, cuando inicia un publicador, recibe la última publicación, publicada antes de que se inicie el suscriptor, como primera publicación coincidente. La alternativa es proporcionar a los suscriptores las publicaciones publicadas únicamente después de que se haya iniciado el suscriptor. De manera adicional, un suscriptor tiene la opción de declinar la recepción de una publicación retenida especificando `MQSO_NEW_PUBLICATIONS_ONLY` en su suscripción.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
Añada 1 a la longitud de la serie pasada a `MQPUT` para pasar el carácter de terminación nula a IBM MQ como parte del almacenamiento intermedio de mensajes.

¿Qué demuestra el primer ejemplo? El ejemplo imita al máximo el patrón tradicional probado para escribir programas de IBM MQ de punto a punto. Una característica importante del patrón de programación de IBM MQ es que el programador no debe preocuparse por dónde se envían los mensajes. La tarea del programador es conectarse a un gestor de colas y pasarle los mensajes que se van a distribuir en los destinatarios. En el paradigma punto a punto, el programador abre una cola (probablemente una cola alias) que el administrador ha configurado. La cola alias direcciona los mensajes a una cola de destino, ya sea en el gestor de colas local o en un gestor de colas remoto. Mientras los mensajes esperan para entregarse, se almacenan en colas en algún lugar entre el origen y el destino.

En el patrón de publicación/suscripción, en lugar de abrir una cola, el programador abre un tema. En nuestro ejemplo, un administrador asocia el tema con una serie de tema. El gestor de colas reenvía la publicación, utilizando colas, a los suscriptores locales o remotos que tienen suscripciones que coinciden con la serie de tema de la publicación. Si las publicaciones se retienen, el gestor de colas guarda la última copia de la publicación, aunque no tenga suscriptores ahora. La publicación retenida está disponible para reenviarse a futuros suscriptores. La aplicación de publicador no desempeña ningún papel en la selección o el direccionamiento de la publicación al destino; su tarea es crear y poner publicaciones en los temas definidos por el administrador.

El ejemplo de tema fijo es atípico de muchas aplicaciones de publicación/suscripción: es estático. Requiere que el administrador defina las series de tema y cambie los temas en las que se publican. Normalmente, las aplicaciones de publicación/suscripción deben conocer parte o el árbol de temas completo. Quizás los temas cambian con frecuencia, o quizás aunque los temas no cambian mucho, el número de combinaciones de temas es grande y es demasiado oneroso para que un administrador pueda definir un nodo de tema para cada serie de tema donde deba publicarse. Quizás las series de tema no se conocen antes de la publicación; o una aplicación de publicador puede utilizar información del contenido de la publicación para especificar una serie de tema, o puede que tenga información sobre las series de tema donde deben publicarse desde otro origen, por ejemplo, una entrada humana en un navegador. Para cubrir las necesidades de estilos de publicación más dinámicos, el siguiente ejemplo muestra cómo crear temas dinámicamente, como parte de la aplicación de publicador.

Los temas emparejan publicadores y suscriptores. El diseño de reglas o la arquitectura para nombrar temas y organizarlos en árboles de temas es un paso importante en el desarrollo de una solución de publicación/suscripción. Observe atentamente el grado con el que la organización del árbol de temas enlaza los programas de publicador y suscriptor, y los enlaza al contenido del árbol de temas. Pregúntese si los cambios en el árbol de temas afectan a las aplicaciones de publicador y suscriptor, y cómo puede minimizar el efecto. Incorporada en la arquitectura del modelo de publicación/suscripción de IBM MQ se encuentra la noción de un objeto de tema administrativo que proporciona la parte raíz o el subárbol raíz de un tema. El objeto de tema da la opción de definir la parte raíz del árbol de temas administrativamente, que simplifica la programación de aplicaciones y las operaciones y, como resultado, aumenta la capacidad de mantenimiento. Por ejemplo, si está desplegando varias aplicaciones de publicación/suscripción que tienen árboles de temas aislados, al definir administrativamente la parte raíz del árbol de temas, garantiza el asilamiento de los árboles de temas, aunque no haya coherencia en los convenios de denominación de temas adoptados por las distintas aplicaciones.

En la práctica, las aplicaciones de publicador incluyen desde la utilización exclusiva temas fijos, como en este ejemplo, a temas variables, como en el siguiente. El “Ejemplo 2: aplicación de publicación en un tema variable” en la página 834 también demuestra la combinación del uso de temas y series de tema.

Conceptos relacionados

“Ejemplo 2: aplicación de publicación en un tema variable” en la página 834
Programa de WebSphere MQ que muestra la publicación en un tema definido por programa.

“Escritura de aplicaciones de suscriptor” en la página 837
Iníciese en la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación de IBM MQ que consume mensaje de una cola, una aplicación que crea una suscripción y no requiere conocimiento sobre colas y, finalmente, un ejemplo que utiliza tanto colas como suscripciones.

Ejemplo 2: aplicación de publicación en un tema variable
Programa de WebSphere MQ que muestra la publicación en un tema definido por programa.

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

Consulte el resultado en la Figura 67 en la página 835.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
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);
}
```

Figura 66. Aplicación de publicación simple de IBM MQ en un tema variable.

```
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
```

Figura 67. Resultado de la segunda aplicación de publicación de ejemplo

Observe lo siguiente sobre este ejemplo.

char topicNameDefault[] = "STOCKS";
El nombre de tema predeterminado STOCKS define una parte de la serie de tema. Puede alterar temporalmente este nombre de tema proporcionándolo como primer argumento del programa, o elimine el uso del nombre de tema especificando / como primer parámetro.

char topicString[101] = "IBM/PRICE";
IBM/PRICE es la serie de tema predeterminada. Puede alterar temporalmente esta serie de tema proporcionándola como segundo argumento del programa.

El gestor de colas combina la serie de tema proporcionada por el objeto de tema STOCKS, "NYSE", con la serie de tema proporcionada por el programa "IBM/PRICE" e inserta una "/" entre las dos series de tema. El resultado es la serie de tema resuelta "NYSE/IBM/PRICE". La serie de tema resultante es la misma que la definida en el objeto de tema IBMSTOCKPRICE y tiene exactamente el mismo efecto.

El objeto de tema administrativo asociado con la serie de tema resuelta no es necesariamente el mismo objeto de tema que la aplicación de publicación ha pasado a MQOPEN. IBM MQ utiliza el árbol implícito en la serie de tema resuelta para determinar qué objeto de tema administrativo define los atributos asociados a la publicación.

Supongamos que hay dos objetos de tema A y B, y A define el tema "a"y B define el tema "a/b" (Figura 68 en la página 836). Si el programa publicador hace referencia al objeto de tema A y proporciona la serie de tema "b", resolviendo el tema en la serie de tema "a/b", la publicación hereda sus propiedades del objeto de tema B porque el tema coincide con la serie de tema "a/b" definida para B.

if (strcmp(argv[1],"/"))
argv[1] es el nombre de tema opcional proporcionado. "/" no es válido como nombre de tema. Aquí significa que no existe ningún nombre de tema y la serie de tema completa es proporcionada por el programa. El resultado que se muestra en la Figura 67 en la página 835 ilustra cómo el programa proporciona dinámicamente la serie de tema completa.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
En el caso predeterminado, es necesario crear de antemano el elemento opcional topicName, utilizando IBM MQ Explorer o este mandato de MQSC:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
La serie de tema es un campo MQCHARV contenido en el descriptor de tema

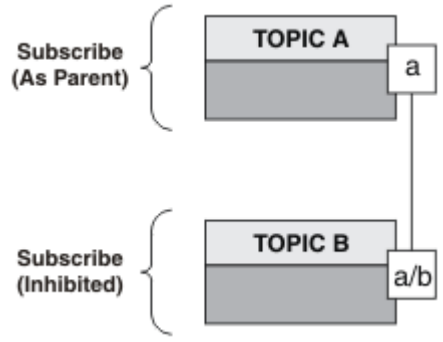


Figura 68. Asociaciones de objeto de tema

¿Qué demuestra el segundo ejemplo? Aunque el código es muy similar al primer ejemplo (efectivamente, sólo hay dos líneas de diferencia), el resultado es un programa significativamente diferente del primero. El programador controla los destinos a los que se envían las publicaciones. Además de la mínima

intervención del administrador para diseñar aplicaciones de suscriptor, no es necesario predefinir temas ni colas para enviar publicaciones desde publicadores a suscriptores.

En el paradigma de la mensajería punto a punto, es necesario definir colas para que los mensajes puedan fluir. IBM MQ implementa la publicación/suscripción utilizando su sistema de gestión de colas subyacente. Las ventajas de la entrega garantizada, transaccionalidad y acoplamiento dinámico asociadas a la gestión de mensajes y colas son heredadas por las publicaciones de publicación/suscripción.

El diseñador de aplicaciones debe determinar si los programas de publicación y suscripción deben reconocer o no el árbol de temas subyacente, y también si los programas de suscripción reconocen la gestión de colas o no. A continuación examine las aplicación de suscripción de ejemplo. Están diseñadas para ser utilizadas con los programas de publicación de ejemplo, normalmente realizando la publicación y suscripción en `NYSE/IBM/PRICE`.

Conceptos relacionados

"Ejemplo 1: Publicador en un tema fijo" en la página 831
Un programa de IBM MQ que ilustra la publicación en un tema definido administrativamente.

"Escritura de aplicaciones de suscriptor" en la página 837
Iníciese en la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación de IBM MQ que consume mensaje de una cola, una aplicación que crea una suscripción y no requiere conocimiento sobre colas y, finalmente, un ejemplo que utiliza tanto colas como suscripciones.

Escritura de aplicaciones de suscriptor

Iníciese en la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación de IBM MQ que consume mensaje de una cola, una aplicación que crea una suscripción y no requiere conocimiento sobre colas y, finalmente, un ejemplo que utiliza tanto colas como suscripciones.

En Tabla 122 en la página 837 aparecen los tres estilos de consumidor o suscriptor, junto con las secuencias de llamadas a funciones de IBM MQ que los caracterizan.

1. El primer estilo, MQ Publication Consumer, es idéntico a un programa MQ de punto a punto que solo realiza la operación MQGET. La aplicación no es consciente de que consume publicaciones, simplemente lee mensajes de una cola. La suscripción que provoca que las publicaciones se dirijan a la cola se crea de forma administrativa utilizando IBM MQ Explorer o un mandato.
2. El segundo estilo es el patrón preferido por la mayoría de las aplicaciones de suscriptor. La aplicación de suscriptor crea la suscripción y luego obtiene publicaciones. La gestión de colas la lleva a cabo enteramente el gestor de colas. Esto se conoce como *suscriptor gestionado*.
3. En el tercer estilo, la aplicación de suscriptor es responsable de especificar la cola que se utilizará para retener publicaciones, abrir y cerrar dicha cola y emitir suscripciones para llenar la cola con publicaciones. Esto se conoce como *suscriptor no gestionado*.

Una manera de entender estos estilos es estudiar los programas `C` de ejemplo que aparecen en Tabla 122 en la página 837 para cada uno de los estilos. Los ejemplos están diseñados para que se ejecuten junto con el ejemplo de publicador que se encuentra en "Escribir aplicaciones de publicación" en la página 830.

Tabla 122. Patrones de programa IBM MQ punto a punto frente a suscripción

Paso	**Consumidor de mensajes MQ**	**"Ejemplo 1: consumidor de Publicación MQ" en la página 838**	**"Ejemplo 2: Suscriptor MQ no gestionado" en la página 840**	**"Ejemplo 3: Suscriptor MQ no gestionado" en la página 845**
Conectarse a un gestor de colas	MQCONN	MQCONN	MQCONN	MQCONN
Abrir cola	MQOPEN	MQOPEN		MQOPEN
Suscribir			MQSUB	MQSUB
Obtener mensajes	MQGET	MQGET	MQGET	MQGET

Tabla 122. Patrones de programa IBM MQ punto a punto frente a suscripción (continuación)				
Paso	**Consumidor de mensajes MQ**	**"Ejemplo 1: consumidor de Publicación MQ" en la página 838**	**"Ejemplo 2: Suscriptor MQ no gestionado" en la página 840**	**"Ejemplo 3: Suscriptor MQ no gestionado" en la página 845**
Cerrar cola	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Cerrar suscripción			MQCLOSE	MQCLOSE
Desconectarse del gestor de colas	MQDISC	MQDISC	MQDISC	MQDISC

El uso de MQCLOSE siempre es opcional, ya sea para liberar recursos, pasar opciones de MQCLOSE o simplemente por simetría con MQOPEN. Dado que no es probable que necesite especificar las opciones de MQCLOSE cuando se cierre la cola de suscripción en el caso del suscriptor MQ gestionado, y el argumento symmetry (simetría) no es relevante, la cola de suscripción no se cierra explícitamente en Ejemplo 2: suscriptor MQ gestionado.

Otra forma de entender los patrones de aplicación de publicación/suscripción es observar las interacciones entre las distintas entidades implicadas. La línea de vida o los diagramas de secuencias UML son una buena manera de estudiar las interacciones. En "Ciclos de vida de publicación/suscripción" en la página 854 se describen tres ejemplos de línea de vida.

Ejemplo 1: consumidor de Publicación MQ
El consumidor de Publicación MQ es un consumidor de mensajes de IBM MQ que no se suscribe a sí mismo a los temas.

Para crear la cola de suscripción y publicación para este ejemplo, ejecute los siguientes mandatos o defina los objetos utilizando IBM MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

La suscripción IBMSTOCKPRICESUB hace referencia al objeto de tema IBMSTOCK creado para el ejemplo de publicador y la cola local STOCKTICKER. El objeto de tema IBMSTOCK define la serie de tema que se utiliza en la suscripción, NYSE/IBM/PRICE. Tenga en cuenta que el objeto de tema y la cola utilizados para recibir las publicaciones deben definirse antes de crear la suscripción.

Hay varias facetas de gran valor para el patrón de consumidor de Publicación MQ:

1. Multiproceso: compartición del trabajo de lectura de publicaciones. Las publicaciones van todas a la única cola asociada al tema de la suscripción. Varios consumidores pueden abrir la cola utilizando MQOO_INPUT_SHARED.
2. Suscripciones gestionadas centralmente. Las aplicaciones no construyen sus propios temas de suscripción ni suscripciones; el administrador es responsable de dónde se envían las publicaciones.
3. Concentración de suscripciones: varias suscripciones diferentes pueden enviarse a una única cola.
4. Duración de la suscripción: la cola recibe todas las publicaciones tanto si hay consumidores activos como si no.
5. Migración y coexistencia: el código de consumidor funciona igualmente bien para un escenario de punto a punto y un escenario de publicación/suscripción.

La suscripción crea una relación entre la serie de tema NYSE/IBM/PRICE y la cola STOCKTICKER. Las publicaciones, incluida cualquier publicación retenida actualmente, se reenvían a STOCKTICKER desde el momento en el que se crea la suscripción.

Una suscripción creada administrativamente puede estar gestionada o no gestionada. Una suscripción gestionada entra en vigor en cuanto se crea, al igual que una suscripción no gestionada. No todas las

facetas de patrón están disponibles para una suscripción gestionada. Consulte también el apartado “Ejemplo 3: Suscriptor MQ no gestionado” en la página 845

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

Los resultados se muestran en Figura 70 en la página 840.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){          /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
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);
}
```

Figura 69. Consumidor de Publicación MQ.

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

Figura 70. Salida del consumidor de Publicación MQ

Hay un par de sugerencias de programación de lenguaje IBM MQ *C* estándar para tener en cuenta:

memset(publication, 0, sizeof(publicationBuffer));
: Asegúrese de que el mensaje tenga un nulo al final para facilitar el formateo utilizando `printf`. El ejemplo de publicador incluye el nulo al final en el almacenamiento intermedio de mensaje pasado al MQPUT mediante la adición de 1 a `strlen(publication)`. El establecimiento de los almacenamientos intermedios de MQCHAR en nulo es un buen estilo de programación para los programas C de IBM MQ C que utilizan los almacenamientos intermedios para almacenar series, lo que garantiza que aparezca un nulo después de una matriz de caracteres que no llena completamente el almacenamiento intermedio.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
: Reserve un nulo al final del almacenamiento intermedio de mensaje para garantizar que el mensaje devuelto tenga un nulo al final en caso de que `if (messlen == strlen(publication));` sea true. Esa sugerencia complementa la anterior y garantiza que haya al menos un nulo en `publicationBuffer` que no se altere temporalmente con el contenido de `publication`.

Conceptos relacionados

“Ejemplo 2: Suscriptor MQ no gestionado” en la página 840
El suscriptor MQ gestionado es el patrón preferido para muchas aplicaciones de suscriptor. Una suscripción gestionada es aquella en la que IBM MQ maneja la suscripción y realiza el registro y la anulación del registro. El ejemplo *no* requiere ninguna definición administrativa de colas, temas o suscripciones.

“Ejemplo 3: Suscriptor MQ no gestionado” en la página 845
El suscriptor no gestionado es una clase importante de aplicación de suscriptor. Con él, puede combinar las ventajas de la publicación/suscripción con el *control* de las colas y el consumo de publicaciones. Una suscripción no gestionada es donde la aplicación es responsable. para especificar la cola donde se almacenan las suscripciones. El ejemplo muestra distintas formas de combinar suscripciones y colas.

“Escribir aplicaciones de publicación” en la página 830
Comience a escribir aplicaciones de publicación estudiando dos ejemplos. El primero se ha diseñado para que sea lo más aproximado posible a una aplicación de punto a punto que transfiere mensajes a una cola y el segundo muestra cómo crear temas de forma dinámica, un patrón más común en las aplicaciones de publicación.

Ejemplo 2: Suscriptor MQ no gestionado

El suscriptor MQ gestionado es el patrón preferido para muchas aplicaciones de suscriptor. Una suscripción gestionada es aquella en la que IBM MQ maneja la suscripción y realiza el registro y la anulación del registro. El ejemplo *no* requiere ninguna definición administrativa de colas, temas o suscripciones.

Este es el tipo de suscriptor gestionado más sencillo que normalmente utiliza una suscripción *no duradera*. El ejemplo está pensado para una suscripción no duradera. La suscripción tan solo dura lo que dura la vida útil del manejador de suscripción de MQSUB. Cualquier publicación que coincida con la serie de tema durante la vida útil de la suscripción se envía a la cola de suscripción (y probablemente se retiene una publicación, si el distintivo `MQSO_NEW_PUBLICATIONS_ONLY` o tiene los valores predeterminados, si se ha retenido una publicación anterior que coincide con la serie de tema y la publicación era persistente, o si no ha finalizado el gestor de colas desde que se ha creado la publicación).

También puede utilizar una suscripción *duradera* con este patrón. Normalmente, una suscripción duradera gestionada se utiliza por motivos de fiabilidad, en lugar de establecer una suscripción que, aunque no ocurra ningún error, sobreviva al suscriptor. Para obtener más información acerca de los ciclos

de vida asociados a las suscripciones gestionadas, no gestionadas, duraderas y no duraderas, consulte la sección de temas relacionados.

A menudo, las suscripciones duraderas están asociadas a las publicaciones persistentes y las suscripciones no duraderas con publicaciones no persistentes, pero no es necesaria la relación entre la duración de la suscripción y la persistencia de la publicación. Son posibles las cuatro combinaciones de persistencia y duración.

En el caso de la combinación gestionada y no duradera en cuestión, el gestor de colas crear una cola de suscripción que se depura y suprime cuando se cierra la cola. Las publicaciones se eliminan de la cola cuando se cierra la suscripción no duradera.

El patrón gestionado no duradero de ejemplo que muestra este código tiene las ventajas siguientes:

1. Suscripción a petición: La serie de tema de suscripción es dinámica. La proporciona la aplicación cuando se ejecuta.
2. Cola autogestionada: La cola de suscripción se autodefine y autogestiona.
3. Ciclo de vida de suscripción autogestionada: Las suscripciones *no duraderas* solo existen mientras dura la aplicación de suscriptor.
   - Si define una suscripción *duradera* se genera una cola de suscripción permanente y las publicaciones se continúan almacenando en ella sin que ningún programa de suscriptor esté activo. El gestor de colas suprime la cola y borra de la cola cualquier publicación que no se haya recuperado, solo después de que la aplicación o el administrador hayan optado por suprimir la suscripción. La suscripción se puede suprimir con un mandato administrativo o cerrando la suscripción con la opción `MQCO_REMOVE_SUB`.
   - Considere la posibilidad de establecer `SubExpiry` para las suscripciones duraderas, de modo que dejen de enviarse las publicaciones a la cola y el suscriptor pueda consumir cualquier publicación restante antes de eliminar la suscripción y que, en consecuencia, el gestor de colas suprima la cola y cualquier publicación que quede en la misma.
4. Despliegue flexible de temas de suscripción flexible: La gestión de temas de suscripción se simplifica definiendo la parte raíz de la suscripción utilizando un tema definido de forma administrativa. Esto oculta la parte de la raíz del árbol de temas para la aplicación. Al ocultar la parte raíz, se puede desplegar una aplicación sin que ésta cree de forma accidental un árbol de temas que solape otro árbol de temas que pueda haber creado otra instancia o aplicación.
5. Temas administrados: Mediante una serie de temas en la que la primera parte coincide con un objeto de tema definido de forma administrativa, las publicaciones se gestionan según los atributos del objeto de tema.
   - Por ejemplo, si la primera parte de la serie de tema coincide con la serie de tema asociada a un objeto de tema en clúster, la suscripción puede recibir publicaciones de otros miembros del clúster.
   - La coincidencia selectiva de los objetos de tema definidos de forma administrativa y de las suscripciones definidas de forma programada le permite combinar las ventajas de ambos. El administrador proporciona atributos para temas y el programador define dinámicamente los subtemas sin ocuparse de la gestión de los temas.
   - De hecho, se utiliza la serie de tema resultante para la coincidencia del objeto de tema que proporciona los atributos asociados al mismo, y no necesariamente el objeto de tema mencionado en `sd.Objectname`, aunque normalmente acaban siendo exactamente lo mismo. Consulte “Ejemplo 2: aplicación de publicación en un tema variable” en la página 834.

Al crear la suscripción duradera en el ejemplo, se continúan enviando las publicaciones a la cola de suscripción después de que el suscriptor cierra la suscripción con la opción `MQCO_KEEP_SUB`. La cola continúa recibiendo publicaciones cuando el suscriptor está activo. Puede alterar temporalmente este comportamiento creando la suscripción con la opción `MQSO_PUBLICATIONS_ON_REQUEST` y utilizando `MQSUBRQ` para solicitar la publicación retenida.

Se puede reanudar la suscripción posteriormente abriendo la suscripción con la opción `MQCO_RESUME`.

Puede utilizar el manejador de cola, `Hobj`, que devuelve `MQSUB` de varios modos. En el ejemplo, el manejador de cola se utiliza para averiguar el nombre de la cola de suscripción. Las colas gestionadas

se abren utilizando las colas del modelo predeterminado SYSTEM.NDURABLE.MODEL.QUEUE o SYSTEM.DURABLE.MODEL.QUEUE. Puede alterar los valores predeterminados proporcionando sus propias colas de modelos duraderos y no duraderos, para cada tema de forma individual, como propiedades del objeto de tema asociado a la suscripción.

Independientemente de los atributos heredados de las colas de modelos, no puede reutilizar un manejador de cola gestionado para crear una suscripción adicional. Ni tampoco puede obtener otro manejador para la cola gestionada abriendo por segunda vez la cola gestionada mediante el nombre de cola que se ha devuelto. La cola se comporta como si se hubiera abierto para entrada exclusiva.

Las colas no gestionadas son más flexibles que las colas gestionadas. Por ejemplo, puede compartir colas no gestionadas o definir varias suscripciones en la cola individual. El ejemplo siguiente muestra cómo combinar suscripciones con una cola de suscripción no gestionada.

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

Los resultados se muestran en Figura 73 en la página 844.

```
#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);
}
```

Figura 71. Suscriptor MQ gestionado - parte 1: declaraciones y manejo de parámetros.

Se incluyen algunos comentarios adicionales sobre el código de este ejemplo.

MQHOBJ Hobj = MQHO_NONE;
No puede abrir de forma explícita una cola de suscripción gestionada no duradera para que reciba publicaciones, pero debe asignar almacenamiento para el descriptor de contexto de objeto que devuelve el gestor de colas cuando abre la cola de forma automática. Es importante inicializar el descriptor para MQHO_OBJECT. Esto indica al gestor de colas que debe devolver un descriptor de cola a la cola de suscripción.

MQSD sd = {MQSD_DEFAULT};
El nuevo descriptor de suscripción utilizado en MQSUB.

MQCHAR48 qName;
Aunque el ejemplo no requiere que se conozca la cola de suscripción, el ejemplo no consulta el nombre de la cola de suscripción. En el lenguaje C, el enlace MQINQ es un poco extraño, por lo que le resultará útil estudiar esta parte del ejemplo.

```
    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;
}
```

Figura 72. Suscriptor MQ gestionado - parte 2: cuerpo del código.

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.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.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

Figura 73. MQ subscriber

Se incluyen algunos comentarios adicionales sobre el código de este ejemplo.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Si topicName es nulo o está en blanco *(valor predeterminado)*, no se utiliza el nombre de tema para calcular la serie de tema no resuelta.

sd.ObjectString.VSPtr = topicString;
En lugar de utilizar únicamente un objeto de tema predefinido, en este ejemplo el programador proporciona un objeto de tema y una serie de tema que se combinan mediante MQSUB. Tenga en cuenta que la serie de tema es una estructura MQCHARV.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Una alternativa para establecer la longitud de un campo MQCHARV.

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Después de definir la serie de tema, se debe prestar mucha atención a los distintivos sd.Options. Hay muchas opciones y el ejemplo solo especifica las que más se utilizan comúnmente. Las otras opciones utilizan los valores predeterminados.

1. Dado que la suscripción es *no duradera*, es decir, tiene la vida útil de la suscripción en la aplicación, establezca el distintivo MQSO_CREATE. También puede establecer el distintivo *(valor predeterminado)* MQSO_NON_DURABLE para facilitar la lectura.
2. MQSO_CREATE se complementa con MQSO_RESUME. Ambos distintivos se pueden establecer conjuntamente. El gestor de colas crea una nueva suscripción o reanuda una suscripción existente, según resulte adecuado. No obstante, si especifica MQSO_RESUME también debe inicializar la estructura MQCHARV para sd.SubName, incluso si no hay ninguna suscripción que reanudar. Si no se inicializa SubName se genera un código de retorno 2440: MQRC_SUB_NAME_ERROR desde MQSUB.

   Nota: MQSO_RESUME siempre se omite para una suscripción gestionada no duradera: pero si se especifica sin inicializar la estructura MQCHARV para sd.SubName se genera el error.
3. Además existe un tercer distintivo que afecta al modo en que se abre la suscripción, MQSO_ALTER. En función de los permisos correctos, se modifican las propiedades de la suscripción reanudada de modo que coincidan con los otros atributos especificados en MQSUB.

   Nota: Se debe especificar al menos uno de los atributos MQSO_CREATE, MQSO_RESUME y MQSO_ALTER. Consulte la sección Opciones (MQLONG). Hay tres ejemplos de uso de los tres distintivos en la sección “Ejemplo 3: Suscriptor MQ no gestionado” en la página 845.
4. Establezca MQSO_MANAGED para que el gestor de colas gestione automáticamente la suscripción.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Opcionalmente, omita la definición de la longitud de MQCHARV para las series terminadas en nulos y, en su lugar, utilice el distintivo terminador de nulos.

sd.ResObjectString.VSPtr = resTopicStr;
La serie de tema resultante se repite en el primer printf del programa. Configure MQCHARV ResObjectString para que IBM MQ devuelve la serie resuelta al programa.

Nota: `resTopicStringBuffer` se inicializa en nulos en `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Las series de tema no devueltas no finalizan con un nulo de cola.

`sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;`
Establezca el tamaño de almacenamiento de `sd.ResObjectString` en un número menos que su tamaño real. De este modo, no se sobrescribe el terminador de nulos proporcionado, en caso de que la serie de tema resuelta rellene el almacenamiento intermedio completo.

Nota: No se devuelve ningún error si la serie de tema tiene una longitud mayor que `sizeof(resTopicStrBuffer)-1`. Incluso si `VSLength > VSBufSiz`, la longitud devuelta en `sd.ResObjectString.VSLength` es la longitud de la serie completa y no necesariamente la longitud de la serie devuelta. Pruebe `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` para confirmar que la serie de tema se ha completado.

`MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);`
La función `MQSUB` crea una suscripción. Si es no duradera, probablemente no esté interesado en su nombre, aunque puede averiguar su estado en IBM MQ Explorer. Puede proporcionar el parámetro `sd.SubName` como entrada, para saber el nombre que busca. Por razones obvias, debe evitar conflictos de nombres con otras suscripciones.

`MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);`
Cerrar tanto la suscripción como la cola de suscripción es opcional. En el ejemplo, la suscripción está cerrada pero así la cola. De todos modos, en este caso la opción `MQCLOSE` `MQCO_REMOVE_SUB` es el valor predeterminado ya que la suscripción es no duradera. Utilizar `MQCO_KEEP_SUB` es un error.

Nota: La *cola* no la cierra `MQSUB` y su manejador, `Hobj`, continúa siendo válido hasta que se cierra la cola mediante `MQCLOSE` o `MQDISC`. Si la aplicación finaliza de forma prematura, el gestor de colas limpia la cola y la suscripción una finalizada la aplicación.

Conceptos relacionados

“Ejemplo 1: consumidor de Publicación MQ” en la página 838
El consumidor de Publicación MQ es un consumidor de mensajes de IBM MQ que no se suscribe a sí mismo a los temas.

“Ejemplo 3: Suscriptor MQ no gestionado” en la página 845
El suscriptor no gestionado es una clase importante de aplicación de suscriptor. Con él, puede combinar las ventajas de la publicación/suscripción con el *control* de las colas y el consumo de publicaciones. Una suscripción no gestionada es donde la aplicación es responsable. para especificar la cola donde se almacenan las suscripciones. El ejemplo muestra distintas formas de combinar suscripciones y colas.

“Escribir aplicaciones de publicación” en la página 830
Comience a escribir aplicaciones de publicación estudiando dos ejemplos. El primero se ha diseñado para que sea lo más aproximado posible a una aplicación de punto a punto que transfiere mensajes a una cola y el segundo muestra cómo crear temas de forma dinámica, un patrón más común en las aplicaciones de publicación.

Ejemplo 3: Suscriptor MQ no gestionado
El suscriptor no gestionado es una clase importante de aplicación de suscriptor. Con él, puede combinar las ventajas de la publicación/suscripción con el *control* de las colas y el consumo de publicaciones. Una suscripción no gestionada es donde la aplicación es responsable. para especificar la cola donde se almacenan las suscripciones. El ejemplo muestra distintas formas de combinar suscripciones y colas.

El patrón no gestionado se asocia más frecuentemente con suscripciones *duraderas* que con *no duraderas*. Normalmente, el ciclo de vida de una suscripción creada por un suscriptor no gestionado es independiente del ciclo de vida de la propia aplicación de suscripción. Al hacer que la suscripción sea duradera, la suscripción recibe publicaciones incluso cuando no hay ninguna aplicación de suscripción activa.

Puede crear suscripciones *gestionadas* duraderas para conseguir el mismo resultado, pero algunas aplicaciones requieren una mayor flexibilidad y control sobre las colas y los mensajes de lo que es posible con una suscripción gestionada. Para una suscripción gestionada duradera, el gestor de colas crea una cola permanente para las publicaciones que coinciden con el tema de la suscripción. Suprime la cola y las publicaciones asociadas cuando se suprime la suscripción.

Normalmente, se utilizan suscripciones *gestionadas* duraderas si el ciclo de vida de la aplicación y la suscripción es esencialmente el mismo, pero difícil de garantizar. Al hacer que la suscripción sea duradera, y que el publicador cree publicaciones persistentes, no hay pérdida de mensajes si el gestor de colas o el suscriptor termina prematuramente y debe recuperarse.

Para aplicaciones no JMS o aplicaciones JMS que no utilizan una suscripción compartida, el gestor de colas abrirá implícitamente la cola de suscripción gestionada duradera para un suscriptor de tal forma que el proceso compartido de la cola no sea posible. Además, a menos que la aplicación esté utilizando suscripciones compartidas JMS, no es posible crear más de una suscripción para cada cola gestionada y es posible que le resulte más difícil gestionar las colas porque tiene menos control sobre los nombres de las colas. Por estas razones, considere si el suscriptor MQ *no gestionado* es una opción más apropiada para las aplicaciones que requieren suscripciones duraderas que el suscriptor MQ *gestionado*.

El código de la Figura 76 en la página 851 muestra un patrón de suscripción duradera no gestionada. A título ilustrativo, el código también crea suscripciones no duraderas no gestionadas. Este ejemplo ilustra las siguientes facetas de patrón:

- Suscripciones a petición: las series de tema de suscripción son dinámicas. Las proporciona la aplicación cuando se ejecuta.
- Gestión simplificada de temas de suscripción: la gestión de temas de suscripción se ha simplificado mediante la definición de la parte de raíz de la serie de tema de suscripción utilizando un tema definido administrativamente. Esto oculta la parte de raíz del árbol de temas a la aplicación. Al ocultar la parte de raíz, se puede desplegar un suscriptor en distintos árboles de temas.
- Gestión de suscripciones flexible: puede definir una suscripción administrativamente o crearla bajo demanda en un programa de suscriptor. No hay ninguna diferencia entre las suscripciones creadas administrativamente y mediante programación, excepto un atributo que muestra cómo se ha creado la suscripción. Hay un tercer tipo de suscripción que el gestor de colas crea automáticamente para la distribución de suscripciones. Se muestran todas las suscripciones en IBM MQ Explorer.
- Asociación flexible de suscripciones a colas: la función MQSUB asocia una cola local predefinida a una suscripción. Hay diferentes maneras de utilizar MQSUB para asociar suscripciones a colas:
  - Asocie una suscripción con una cola que *no* tenga suscripciones existentes, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Asocie una *nueva* suscripción con una cola que tenga suscripciones existentes, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Mueva una suscripción existente a una cola diferente, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Reanude una suscripción existente asociada con una cola existente, `MQSO_RESUME + (Hobj = MQHO_NONE)` o `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Combinando `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` en diferentes combinaciones, puede cubrir diferentes estados de entrada de la suscripción y la cola sin tener que codificar varias versiones de MQSUB con diferentes valores de `sd.Options`.
  - De forma alternativa, si codifica una opción específica de `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` el gestor de colas devuelve un error (Tabla 123 en la página 848) si los estados de la suscripción y la cola proporcionados como entrada para MQSUB son incoherentes con el valor de `sd.Options`. La Figura 82 en la página 854 muestra los resultados de emitir MQSUB para la Suscripción X con diferentes valores individuales del distintivo `sd.Options`, y de pasar tres descriptores de contexto del objeto diferentes.

  Explore las diferentes entradas para el programa de ejemplo en la Figura 75 en la página 850 para familiarizarse con estos diferentes tipos de errores. Un error común, `RC = 2440`, que no está incluido en los casos listados en la tabla, es un error de nombre de suscripción. La causa suele ser pasar un nombre de suscripción nulo o no válido con `MQSO_RESUME` o `MQSO_ALTER`.
- Multiproceso: Puede compartir el trabajo de lectura de publicaciones con muchos consumidores. Las publicaciones van todas a la única cola asociada al tema de la suscripción. Los consumidores tienen la opción de abrir la cola directamente utilizando MQOPEN o reanudar la suscripción utilizando MQSUB.

- Concentración de suscripciones: se pueden crear varias suscripciones en la misma cola. Preste atención con esta función, ya que puede solapar las suscripciones y puede recibir la misma publicación varias veces. La opción MQSO_GROUP_SUB elimina las publicaciones duplicadas causadas por el solapamiento de suscripciones.
- Separación de suscriptor y consumidor: además de los tres modelos de consumidor ilustrados en los ejemplos, otro modelo es separar el consumidor del suscriptor. Se trata de una variación del suscriptor MQ gestionado, pero en lugar de emitir MQOPEN y MQSUB en el mismo programa, un programa se suscribe a publicaciones, y otro programa las consume. Por ejemplo, el suscriptor puede ser parte de un clúster de publicación/suscripción y el consumidor esté conectado a un gestor de colas fuera del clúster de gestores de colas. El consumidor recibe publicaciones a través de la gestión de colas distribuidas estándar, definiendo la cola de suscripciones como una definición de cola remota.

Comprender el comportamiento de MQSO_CREATE | MQSO_RESUME | MQSO_ALTER es importante, especialmente si tiene previsto simplificar el código utilizando combinaciones de estas opciones. Estudie la Tabla 123 en la página 848 que muestra los resultados de pasar diferentes descriptores de contexto de cola a MQSUB, y los resultados de ejecutar el programa de ejemplo mostrado en la Figura 77 en la página 852 a la Figura 82 en la página 854.

El escenario utilizado para construir la tabla tiene una suscripción X y dos colas, A y B. El parámetro de nombre de suscripción sd.SubName se establece en X, el nombre de una suscripción conectada a la cola A. La cola B no tiene ninguna suscripción asociada.

En Tabla 123 en la página 848, MQSUB se pasa la suscripción X y el descriptor de contexto de cola a la cola A. Los resultados de las opciones de suscripción son los siguientes:

- MQSO_CREATE falla porque el descriptor de contexto de cola corresponde a la cola A que ya tiene una suscripción a X. Compare este comportamiento con la llamada satisfactoria. Esa llamada se realiza satisfactoriamente porque la cola B no tiene ninguna suscripción a X asociada a ella.
- MQSO_RESUME se ejecuta correctamente porque el descriptor de contexto de cola corresponde a la cola A que ya tiene una suscripción a X. Por el contrario, la llamada falla cuando la suscripción X no existe en la cola A.
- MQSO_ALTER se comporta de forma similar a MQSO_RESUME con respecto a la apertura de la suscripción y la cola. Sin embargo, si los atributos contenidos en el descriptor de suscripción que se pasa a MQSUB difieren de los atributos de la suscripción, MQSO_RESUME falla, mientras que MQSO_ALTER se realiza satisfactoriamente siempre que la instancia de programa tenga permiso para modificar los atributos. Tenga en cuenta que nunca puede cambiar la serie de tema de una suscripción; pero en lugar de devolver un error, MQSUB ignora los valores de nombre de tema y serie de tema en el descriptor de suscripción y utiliza los valores de la suscripción existente.

A continuación, consulte Tabla 123 en la página 848 donde se pasa la suscripción X de MQSUB y el descriptor de contexto de cola a la cola B. Los resultados de las opciones de suscripción son los siguientes:

- MQSO_CREATE se ejecuta correctamente y crea la suscripción X en la cola B, debido a que esta cola es una nueva suscripción en la cola B.
- MQSO_RESUME falla. MQSUB busca la suscripción X en la cola B y no la encuentra, pero en lugar de devolver *RC = 2428 - la suscripción X no existe*, devuelve *RC = 2019 - La cola de suscripción no coincide con el manejador de objetos de cola*. El comportamiento de la tercera opción MQSO_ALTER sugiere la razón de este error inesperado. MQSUB espera que el descriptor de contexto de cola apunte a una cola con una suscripción. Comprueba esto primero antes de comprobar si la suscripción nombrada en sd.SubName existe.
- MQSO_ALTER se realiza satisfactoriamente, y mueve la suscripción de la cola A a la cola B.

Un caso que no se muestra en la tabla es si el nombre de suscripción de la suscripción en la cola A no coincide con el nombre de suscripción en sd.SubName. La llamada falla con *RC = 2428 - la suscripción X no existe en la Cola A*.

Tabla 123. Errores de MQSUB con diferentes combinaciones de suscripciones y descriptores de contexto de cola

Descriptores de contexto de cola	**Cola A** **Suscripción X** **Cola B** **Ninguna suscripción**	**Cola A** **Ninguna suscripción** **Cola B** **Ninguna suscripción**
Hobj para la *Cola A* pasado a MQSUB	**MQSO_CREATE** RC = 2432 - La suscripción X ya existe en la Cola A **MQSO_RESUME** Reanuda la suscripción X en la Cola A **MQSO_ALTER** Reanuda la suscripción X en la Cola A y realiza modificaciones permitidas	**MQSO_CREATE** Crea la suscripción X en la Cola A **MQSO_RESUME** RC = 2428 - La suscripción X no existe en la Cola A **MQSO_ALTER** RC = 2428 - La suscripción X no existe en la Cola A
Hobj para la *Cola B* pasado a MQSUB	**MQSO_CREATE** Crea una nueva suscripción X en la Cola B **MQSO_RESUME** RC = 2019 - La cola de suscripción no coincide con el manejador de objetos de cola **MQSO_ALTER** Mover la suscripción X de la Cola A a la Cola B	**MQSO_CREATE** Crea una nueva suscripción X en la Cola B **MQSO_RESUME** RC = 2428 - la suscripción X no existe en la Cola B **MQSO_ALTER** RC = 2428 - la suscripción X no existe en la Cola B
MQHO_NONE pasado a MQSUB	**MQSO_CREATE** RC = 2019 - Manejador de cola erróneo: Establezca el distintivo MQSO_MANAGED para crear una suscripción de cola gestionada y crear una cola gestionada **MQSO_RESUME** Reanuda la suscripción X en la Cola A y devuelve Hobj a la Cola A **MQSO_ALTER** Reanuda la suscripción X en la Cola A, devuelve Hobj a la Cola A y realiza modificaciones permitidas	**MQSO_CREATE** RC = 2019 - Manejador de cola erróneo: Establezca el distintivo MQSO_MANAGED para crear una suscripción de cola gestionada y crear una cola gestionada **MQSO_RESUME** RC = 2428 - No existe la suscripción X **MQSO_ALTER** RC = 2019 - Manejador de cola erróneo: No hay una cola A o B

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

```
#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_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

Figura 74. Suscriptor MQ no gestionado - parte 1: declaraciones.

```
    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(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

Figura 75. Suscriptor MQ gestionado - parte 2: manejo de parámetros.

Los comentarios adicionales sobre el manejo de parámetros en este ejemplo son los siguientes:

switch((argv[5][0]))
Puede optar por especificar A lter | C reate | R esume en el parámetro 5, para probar el efecto de reemplazar la parte del valor de la opción MQSUB que se utiliza en el ejemplo de forma predeterminada. El valor predeterminado utilizado en el ejemplo es MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Nota: Establecer MQSO_ALTER o MQSO_RESUME sin establecer MQSO_DURABLE es un error, y sd.SubName debe establecerse y hacer referencia a una suscripción que se pueda modificar o reanudar.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Si la cola de suscripciones predeterminada STOCKTICKER se sustituye por una serie vacía, mientras MQSO_CREATE esté establecido, el ejemplo establece el distintivo MQSO_MANAGED y crea una

cola de suscripciones dinámica. Si `Alter or Resume` se establece en el quinto parámetro, el comportamiento del ejemplo dependerá del valor de `subscriptionName`.

`*subscriptionName = '\0';`
`sdOptions = sdOptions - MQSO_DURABLE;`

Si la suscripción predeterminada, `IBMSTOCKPRICESUB`, se sustituye por una serie vacía, entonces el ejemplo elimina el distintivo `MQSO_DURABLE`. Si ejecuta el ejemplo proporcionando los valores predeterminados para los demás parámetros, se crea una suscripción temporal adicional destinada a `STOCKTICKER` y recibe publicaciones duplicadas. La próxima vez que ejecute el ejemplo, sin ningún parámetro, recibirá de nuevo una sola publicación.

```
    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,
&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_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,
&CompCode, &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;
}
```

Figura 76. Suscriptor MQ no gestionado - parte 3: cuerpo del código.

Los comentarios adicionales sobre el código de este ejemplo son los siguientes:

if (strlen(subscriptionQueue))
Si no hay ningún nombre de cola de suscripciones, el ejemplo utiliza MQHO_NONE como el valor de Hobj.

MQOPEN(...);
La cola de suscripción se abre y el descriptor de contexto de cola se guarda en Hobj.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
La suscripción se abre utilizando el Hobj que se pasado desde MQOPEN (o MQHO_NONE si no hay ningún nombre de cola de suscripciones). Una cola no gestionada se puede reanudar sin abrirla explícitamente con una llamada MQOPEN.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
La suscripción se cierra utilizando el descriptor de contexto de suscripción. Dependiendo de si la suscripción es duradera o no, la suscripción se cierra con una MQCO_KEEP_SUB o MQCO_REMOVE_SUB implícita. Puede cerrar una suscripción duradera con MQCO_REMOVE_SUB, pero no puede cerrar una suscripción no duradera con MQCO_KEEP_SUB. La acción de MQCO_REMOVE_SUB es eliminar la suscripción, lo que impide el envío de nuevas publicaciones a la cola de suscripciones.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
No se lleva a cabo ninguna acción especial si la suscripción es no gestionada. Si la cola es gestionada y la suscripción se ha cerrado con una MQCO_REMOVE_SUB explícita o implícita, todas las publicaciones se depuran de la cola y la cola se suprime en este punto.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Asegúrese de que los mensajes recibidos son los mensajes para nuestra suscripción.

Los resultados del ejemplo ilustran aspectos de la publicación/suscripción:

En la Figura 77 en la página 852 el ejemplo comienza con la publicación de 130 en el tema 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
```

Figura 77. Publicar 130 en NYSE/IBM/PRICE

En la ejecución Figura 78 en la página 852 del ejemplo, cuando se utilizan los parámetros predeterminados se recibe la publicación retenida 130. Se hace caso omiso del objeto de tema y la serie de tema proporcionados, tal como se muestra en la Figura 82 en la página 854. El objeto de tema y la serie de tema siempre se toman del objeto de suscripción, cuando se proporciona uno, y la serie de tema es inmutable. El comportamiento real del ejemplo depende de la selección o combinación de MQSO_CREATE, MQSO_RESUME y MQSO_ALTER. En este ejemplo, MQSO_RESUME es la opción seleccionada.

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|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
```

Figura 78. Recibir la publicación retenida

En (Figura 79 en la página 853) no se reciben publicaciones debido a que la suscripción duradera ya ha recibido la publicación retenida. En este ejemplo, la suscripción se reanuda proporcionando únicamente el nombre de suscripción, sin el nombre de cola. Si se ha proporcionado el nombre de cola, la cola se abriría primero y el descriptor de contexto se pasaría a MQSUB.

Nota: El error 2038 de MQINQ es debido a que la MQOPEN implícita de STOCKTICKER mediante MQSUB no incluye la opción MQOO_INQUIRE. Evite el código de retorno 2038 de MQINQ abriendo la cola explícitamente.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|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
```

Figura 79. Reanudar la suscripción

En la Figura 80 en la página 853, el ejemplo crea una suscripción no gestionada no duradera utilizando STOCKTICKER como el destino. Debido a que esta es una nueva suscripción, recibe la publicación retenida.

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|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
```

Figura 80. Recibir la publicación retenida con una nueva suscripción no gestionada no duradera

En la Figura 81 en la página 853, para demostrar el solapamiento de suscripciones, se envía otra publicación, cambiando la publicación retenida. A continuación, se crea una nueva suscripción no gestionada no duradera al no proporcionar un nombre de suscripción. La publicación retenida se recibe dos veces, una para la nueva suscripción, y otra para la suscripción IBMSTOCKPRICESUB duradera que sigue activa en la cola STOCKTICKER. El ejemplo ilustra que es la cola la que tiene suscripciones, y no la aplicación. A pesar de no hacer referencia a la suscripción IBMSTOCKPRICESUB en esta invocación de la aplicación, la aplicación recibe la publicación dos veces: una desde la suscripción duradera que se creó administrativamente, y otra desde la suscripción no duradera creada por la propia aplicación.

```
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(reate)|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
```

Figura 81. Solapamiento de suscripciones

En la Figura 82 en la página 854, el ejemplo demuestra que proporcionar una nueva serie de tema y una suscripción existente no da como resultado una suscripción modificada.

1. En el primer caso, Resume reanuda la suscripción existente, como podría esperar, e ignora la serie de tema cambiada.
2. En el segundo caso, Alter provoca un error, RC = 2510, Topic not alterable.
3. En el tercer ejemplo, Create provoca un error 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
```

Figura 82. Los temas de suscripción no se pueden cambiar

Conceptos relacionados

“Ejemplo 1: consumidor de Publicación MQ” en la página 838
El consumidor de Publicación MQ es un consumidor de mensajes de IBM MQ que no se suscribe a sí mismo a los temas.

“Ejemplo 2: Suscriptor MQ no gestionado” en la página 840
El suscriptor MQ gestionado es el patrón preferido para muchas aplicaciones de suscriptor. Una suscripción gestionada es aquella en la que IBM MQ maneja la suscripción y realiza el registro y la anulación del registro. El ejemplo *no* requiere ninguna definición administrativa de colas, temas o suscripciones.

“Escribir aplicaciones de publicación” en la página 830
Comience a escribir aplicaciones de publicación estudiando dos ejemplos. El primero se ha diseñado para que sea lo más aproximado posible a una aplicación de punto a punto que transfiere mensajes a una cola y el segundo muestra cómo crear temas de forma dinámica, un patrón más común en las aplicaciones de publicación.

Ciclos de vida de publicación/suscripción

Tenga en cuenta los ciclos de vida de temas, suscripciones, suscriptores, publicaciones, publicadores y colas a la hora de diseñar aplicaciones de publicación/suscripción.

El ciclo de vida de un objeto como, por ejemplo, una suscripción, empieza en su creación y termina con su borrado. También puede incluir otros estados y cambios por los que pasa como, por ejemplo, una suspensión temporal, tener temas de niveles superior e superior, caducidad y borrado.

Tradicionalmente, los objetos IBM MQ como, por ejemplo, las colas se han creado administrativamente o mediante programas administrativos que utilizan el formato PCF (Programmable Command Format). La publicación/suscripción es distinta al proporcionar los verbos de API `MQSUB` y `MQCLOSE` para crear y suprimir suscripciones, teniendo el concepto de suscripciones gestionadas, que no solo crean y suprimen colas, sino que también limpian los mensajes no consumidos y tienen asociaciones entre objetos de tema creados administrativamente y cadenas de tema creadas programática o administrativamente.

Esta riqueza funcional satisface una amplia gama de requisitos de publicación/suscripción y también simplifica el diseño de algunos patrones comunes de aplicación de publicación/suscripción. Las suscripciones gestionadas, por ejemplo, simplifican tanto la programación y la administración de una suscripción que está pensada para durar tanto tiempo como el programa que la ha creado. Las suscripciones no gestionadas simplifican la programación en la que hay una conexión menos acoplada entre las publicaciones suscriptoras y publicadoras. Las suscripciones creadas de forma centralizada son útiles cuando el patrón es el de direccionar el tráfico de publicación a los consumidores en función de un modelo centralizado de control, por ejemplo, enviar información de vuelos a puertas automatizadas, mientras que las suscripciones creadas programáticamente se pueden usar si el personal de puerta es responsable de suscribirse a los registros de pasajeros de ese vuelo introduciendo un número de vuelo en una puerta.

En este último ejemplo, una suscripción duradera gestionada puede ser adecuada: gestionada, porque las suscripciones se crean con mucha frecuencia, y tienen un punto final claro cuando se cierra la puerta y la suscripción se puede eliminar mediante programación; duradera, para evitar perder un registro de pasajeros debido a que el programa de suscriptor de la puerta se desactiva por una razón u otra[8]. Para iniciar la publicación de los registros de pasajeros en la puerta, un posible diseño sería que la aplicación de la puerta se suscribiera a los registros de pasajeros utilizando el número de la puerta, y publicar el

evento de apertura de la puerta utilizando el número de la puerta. El publicador responde al evento de apertura de puerta publicando los registros de pasajero, que también podrían ir a otras partes interesadas como, por ejemplo, facturación, registro de vuelos, servicios al cliente y envío de mensajes de texto a los móviles de los pasajeros del número de puerta.

La suscripción gestionada centralizadamente podría usar un modelo no gestionado duradero, direccionando las listas de pasajeros a la puerta mediante una cola predefinida para cada puerta.

Los tres ejemplos siguientes de ciclos de vida de publicación/suscripción ilustran cómo suscriptores gestionados no duraderos, gestionados duraderos y no gestionados duraderos interactúan con suscripciones, temas, colas, publicadores y el gestor de colas, y cómo las responsabilidades se pueden repartir entre la administración y los programas de suscriptor.

Suscriptor gestionado no duradero

Figura 83 en la página 856 muestra una aplicación que crea una suscripción no duradera gestionada, obtiene dos mensajes que se publican en el tema identificado en la suscripción y termina. Las interacciones etiquetadas con una fuente gris en cursiva con flechas de puntos son implícitas.

Hay algunos puntos que señalar.

1. La aplicación crea una suscripción en un tema en el que ya se ha publicado dos veces. Cuando el suscriptor recibe su primera publicación, recibe la *segunda* publicación, que es la publicación retenida actualmente.
2. El gestor de colas crea una cola de suscripción temporal, así como una suscripción para el tema.
3. La suscripción tiene una caducidad. Cuando la suscripción caduca, no se envían más publicaciones sobre el tema a esta suscripción, pero el suscriptor sigue obteniendo los mensajes publicados antes de caducar la suscripción. La caducidad de la publicación no se ve afectada por la caducidad de la suscripción.
4. La cuarta publicación no se coloca en la cola de suscripciones y, por tanto, el último `MQGET` no devuelve una publicación.
5. Aunque el suscriptor cierre la suscripción, no se cierra la conexión con la cola ni con el gestor de colas.
6. El gestor de colas se limpia poco después de terminar la aplicación. Puesto que la suscripción es gestionada y no duradera, la cola de suscripciones se borra.

[8] El editor debe enviar los registros de pasajeros como mensajes persistentes para evitar otros posibles fallos, por supuesto.

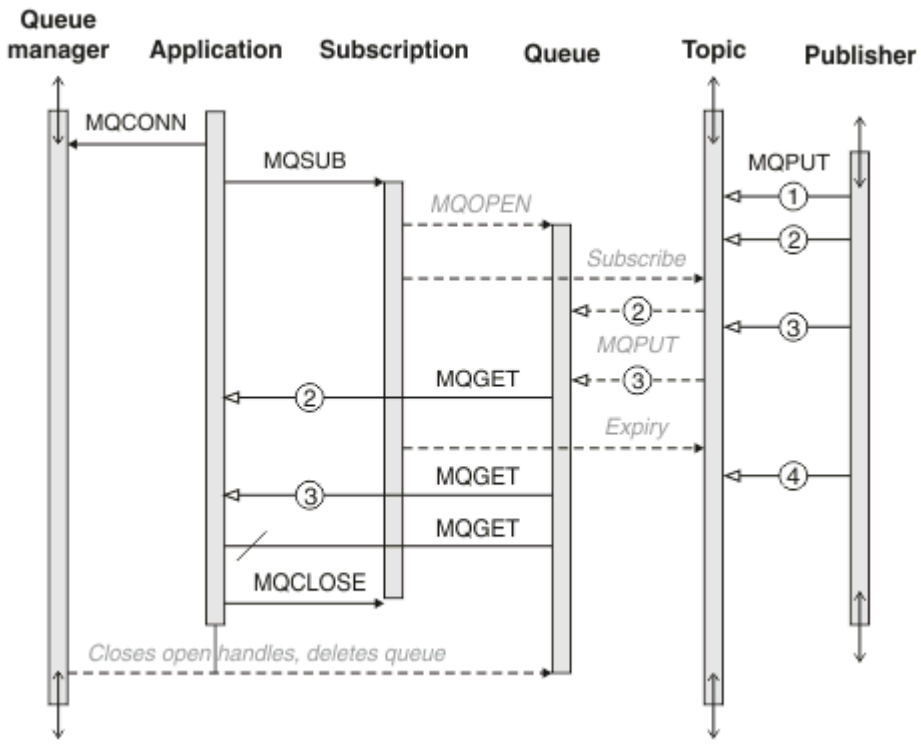


Figura 83. Líneas de vida de un suscriptor gestionado no duradero

Suscriptor gestionado duradero

El suscriptor duradero gestionado va un paso más allá que en el ejemplo anterior y muestra una suscripción gestionada que sobrevive a la terminación y el reinicio de la aplicación de suscripción.

Hay algunos puntos novedosos por señalar.

1. En este ejemplo, a diferencia del último, el tema de publicación no existía antes de ser definido en la suscripción.
2. La primera vez que el suscriptor finaliza, cierra la suscripción con la opción `MQCO_KEEP_SUB`. Ese es el comportamiento predeterminado cuando se cierra implícitamente una suscripción duradera gestionada.
3. Cuando el suscriptor reanuda la suscripción, la cola de suscripciones se vuelve a abrir.
4. La nueva publicación 2, colocada en la cola antes de reabrirse, está disponible a `MQGET`, incluso después de haberse eliminado la suscripción.

   Aunque la suscripción es duradera, el suscriptor recibe de forma fiable todos los mensajes enviados por el publicador solo si la suscripción es duradera *y además* los mensajes son persistentes. La persistencia de los mensajes depende del valor del campo `Persistent` del `MQMD` del mensaje enviado por el publicador. Un suscriptor no tiene control sobre el.
5. Al cerrar la suscripción con el distintivo `MQCO_REMOVE_SUB`, se elimina la suscripción y se para cualquier publicación adicional que se coloque en la cola de suscripciones. Cuando la cola de suscripciones está cerrada, el gestor de colas elimina la publicación no leída 3 y después borra la cola. La acción equivale a borrar de forma administrativa la suscripción.

   Nota: No suprima la cola manualmente, ni emita `MQCLOSE` con la opción `MQCO_DELETE`o `MQCO_PURGE_DELETE`. Los detalles de implementación visibles de una suscripción gestionada no forman parte de la interfaz IBM MQ soportada. El gestor de colas no puede gestionar una suscripción de forma fiable a menos que tenga un control completo.

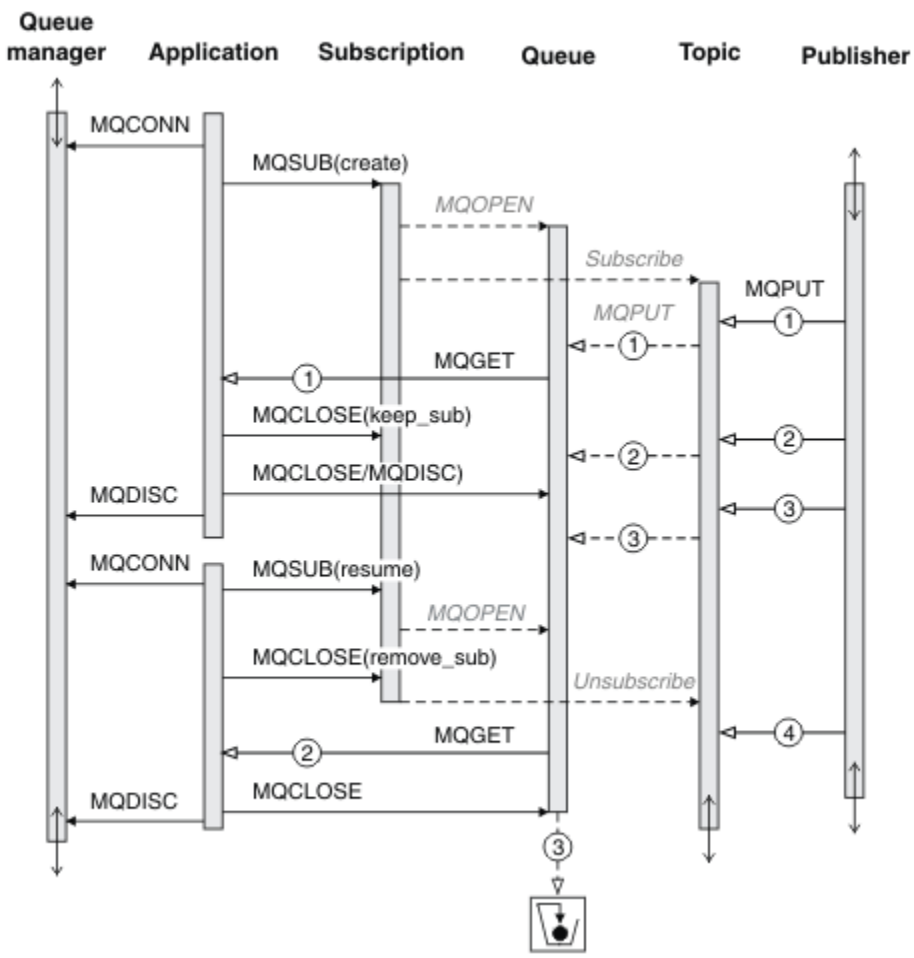


Figura 84. Líneas de vida de un suscriptor duradero gestionado

Suscriptor duradero no gestionado

Se añade un administrador en el tercer ejemplo: el suscriptor duradero no gestionado. Se trata de un buen ejemplo para mostrar cómo el administrador puede interactuar con una aplicación de publicación/suscripción.

Se listan los puntos a tener en cuenta.

1. El publicador coloca un mensaje, 1, en un tema que posteriormente pasa a estar asociado con el objeto de tema que se utiliza en la suscripción. El objeto de tema define una cadena de tema que coincide con el tema que se ha publicado utilizando comodines.
2. El tema tiene una publicación retenida.
3. El administrador crea un objeto de tema, una cola y una suscripción. El objeto de tema y la cola tienen que definirse antes de la suscripción.
4. La aplicación abre la cola asociada a la suscripción y pasa `MQSUB` al manejador de la cola. De forma alternativa, podría simplemente abrir la suscripción, pasándole el manejador de cola `MQHO_NONE`. Lo contrario no se cumple: no puede reanudar una suscripción pasándole únicamente el manejador de cola; una cola podría tener múltiples suscripciones.
5. La aplicación abre la suscripción con la opción `MQSO_RESUME`, aunque es la primera vez que ha abierto la suscripción. Está reanudando una suscripción creada administrativamente.

6. El suscriptor recibe la publicación retenida, 1. La publicación 2, aunque se ha publicado antes de que el suscriptor haya recibido ninguna publicación, se ha publicado después de haberse iniciado la suscripción y es la segunda publicación en la cola de suscripciones.

   Nota: Si la publicación retenida no se ha publicado como un mensaje persistente, se perderá tras reiniciarse el gestor de colas.
7. En este ejemplo, la suscripción es duradera. Un programa tiene la posibilidad de crear una suscripción no duradera no gestionada; tendría que ser obvio que esto no es algo que un administrador pueda hacer.
8. El efecto de la opción MQCO_REMOVE_SUB en el cierre de la suscripción es eliminar la suscripción tal y como si el administrador la hubiera borrado. Esto para cualquier publicación adicional que se envía a la cola, pero no afecta a las publicaciones que ya están en la cola, incluso si se cierra la cola, a diferencia de una suscripción duradera *gestionada*
9. El administrador borra más tarde el mensaje restante, 3 y borra la cola.

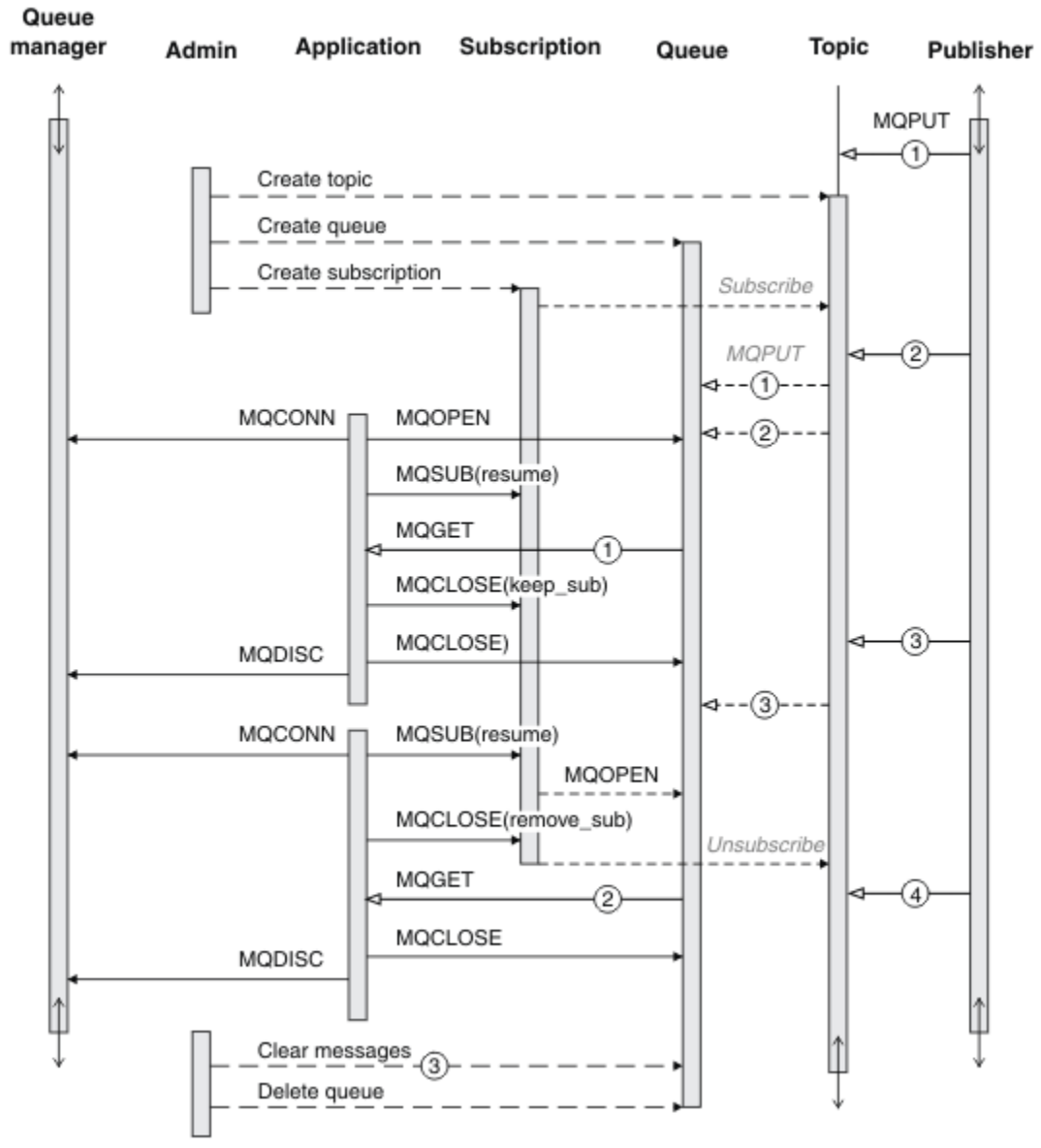


Figura 85. Líneas de vida de un suscriptor duradero no gestionado

Un patrón normal de una suscripción no gestionada consiste en que el administrador realice el mantenimiento de colas y suscripciones. Lo normal es que no se intente emular el comportamiento de un suscriptor gestionado y se limpien las colas y suscripciones programáticamente en el código de la aplicación. Si se ve en la necesidad de escribir la lógica de gestión, plantéese si puede conseguir el mismo resultado utilizando un patrón gestionado. No es fácil escribir un código de gestión plenamente sincronizado y fiable. Resulta más fácil hacer la limpieza después, ya sea manualmente o usando un programa de gestión automático, cuando se puede tener la certeza de que mensajes, suscripciones y colas pueden borrarse sin más, independientemente de su estado.

Propiedades de los mensajes de publicación/suscripción

Varias propiedades de mensaje están relacionadas con los mensajes de publicación/suscripción de IBM MQ .

PubAccountingToken

Este es el valor que se utilizará en el campo AccountingToken del Descriptor de mensaje (MQMD) de todos los mensajes de publicación que coincidan con esta suscripción. AccountingToken forma parte del contexto de identidad del mensaje. Para obtener más información sobre el contexto de mensaje, consulte "Contexto de mensaje" en la página 48. Para obtener más información sobre el campo AccountingToken del MQMD, consulte AccountingToken.

PubApplIdentityData

Este es el valor que se utilizará en el campo ApplIdentityData del Descriptor de mensaje (MQMD) de todos los mensajes de publicación que coincidan con esta suscripción. ApplIdentityData forma parte del contexto de identidad del mensaje. Para obtener más información sobre el contexto de mensaje, consulte "Contexto de mensaje" en la página 48. Para obtener más información sobre el campo ApplIdentityData del MQMD, consulte ApplIdentityData.

Si no se especifica la opción MQSO_SET_IDENTITY_CONTEXT, ApplIdentityData estará en blanco en cada mensaje publicado para esta suscripción, como información de contexto predeterminada.

Si se especifica la opción MQSO_SET_IDENTITY_CONTEXT, el usuario generará PubApplIdentityData y este campo será un campo de entrada que contiene el valor de ApplIdentityData que se debe establecer en cada publicación para esta suscripción.

PubPriority

Este es el valor que se utilizará en el campo Priority del Descriptor de mensaje (MQMD) de todos los mensajes de publicación que coincidan con esta suscripción. Para obtener más información sobre el campo Priority del MQMD, consulte Priority.

El valor debe ser mayor o igual que cero, donde cero es la prioridad más baja. También se pueden utilizar los valores especiales siguientes:

- MQPRI_PRIORITY_AS_Q_DEF- Cuando se proporciona una cola de suscripción en el campo Hobj de la llamada MQSUB, y no es un descriptor de contexto gestionado, la prioridad del mensaje se obtiene del atributo DefProirity de esta cola. Si la cola así identificada es una cola de clúster, o existe más de una definición en la vía de acceso de resolución de nombre de cola, la propiedad se determina cuando el mensaje de publicación se coloca en la cola tal como se describe para Priority en el MQMD. Si la llamada MQSUB utiliza un descriptor de contexto gestionado, la prioridad del mensaje se obtiene del atributo DefPriority de la cola modelo asociada al tema al que está suscrita.
- MQPRI_PRIORITY_AS_PUBLISHED- La prioridad del mensaje es la prioridad de la publicación original. Este es el valor inicial de este campo.

SubCorrelId

Atención: un identificador de correlación sólo se puede pasar entre gestores de colas en un clúster de publicación/suscripción, no en una jerarquía.

Todas las publicaciones enviadas para ser comparadas con esta suscripción contendrán este identificador de correlación en el descriptor de mensaje. Si varias suscripciones utilizan la misma cola para obtener sus publicaciones, el uso de MQGET por ID de correlación sólo permite obtener publicaciones para una suscripción específica. Este identificador de correlación puede ser generado por el gestor de colas o por el usuario.

Si no se especifica la opción MQSO_SET_CORREL_ID, el identificador de correlación es generado por el gestor de colas y este campo será un campo de salida que contiene el identificador de correlación que se establecerá en cada mensaje publicado para esta suscripción.

Si se especifica la opción MQSO_SET_CORREL_ID, el identificador de correlación es generado por el usuario y este campo es un campo de entrada que contiene el identificador de correlación que se debe establecer en cada publicación para esta suscripción. En este caso, si el campo contiene MQCI_NONE, el identificador de correlación que se establecerá en cada mensaje publicado para esta suscripción será el identificador de correlación creado por la colocación original del mensaje.

Si la opción MQSO_GROUP_SUB se ha especificado y el identificador de correlación especificado es el mismo que una suscripción agrupada existente que utiliza la misma cola y una serie de tema que se solapa, sólo la suscripción más significativa del grupo se proporciona con una copia de la publicación.

SubUserData

Estos son los datos de usuario de la suscripción. Los datos proporcionados en la suscripción en este campo se incluirán como la propiedad de mensaje MQSubUserData de cada publicación que se envíe a esta suscripción.

Propiedades de publicación

La Tabla 124 en la página 860 lista las propiedades de publicación que se proporcionan con un mensaje de publicación.

Puede acceder a estas propiedades directamente desde la carpeta **MQRFH2**, o recuperarlas utilizando `MQINQMP`. `MQINQMP` acepta el nombre de propiedad o el nombre de **MQRFH2** como nombre de la propiedad sobre la que se debe obtener información.

Tabla 124. Propiedades de publicación

Nombre de propiedad	Nombre de MQRFH2	Tipo	Descripción
`MQTopicString`	`mqps.Top`	`MQTYPE_STRING`	Serie de tema
`MQSubUserData`	`mqps.Sud`	`MQTYPE_STRING`	Datos de usuario de suscriptor
`MQIsRetained`	`mqps.Ret`	`MQTYPE_BOOLEAN`	Publicación retenida
`MQPubOptions`	`mqps.Pub`	`MQTYPE_INT32`	Opciones de publicación
`MQPubLevel`	`mqps.Pbl`	`MQTYPE_INT32`	Nivel de publicación
`MQPubTime`	`mqpse.Pts`	`MQTYPE_STRING`	Hora de publicación
`MQPubSeqNum`	`mqpse.Seq`	`MQTYPE_INT32`	Número de secuencia de publicación
`MQPubStrIntData`	`mqpse.Sid`	`MQTYPE_STRING`	Datos de tipo serie/entero añadidos por la aplicación de publicación

Tabla 124. Propiedades de publicación (continuación)			
Nombre de propiedad	**Nombre de MQRFH2**	**Tipo**	**Descripción**
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	Formato de mensaje: MQRFH1 MQRFH2 PCF

Orden de los mensajes

Para un tema concreto, el gestor de colas publica los mensajes en el mismo orden en que los recibe de las aplicaciones de publicación, sujetos a un cambio de orden en función de la prioridad de los mensajes.

Normalmente, el orden de los mensajes significa que cada suscriptor recibe mensajes de un gestor de colas concreto, sobre un tema concreto, de una aplicación de publicación en el orden en que dicha aplicación los publica.

No obstante, al igual que con todos los mensajes de IBM MQ, es posible que ocasionalmente los mensajes se entreguen sin orden. Esto puede ocurrir en las situaciones siguientes:

- Si un enlace de red se desactiva y los mensajes posteriores se redirigen a otro enlace
- Si una cola pasa a estar llena temporalmente o inhibida para transferencias, de modo que un mensaje se coloca en una cola de mensajes no entregados, mientras que los mensajes posteriores se transfieren directamente.
- Si el administrador suprime un gestor de colas cuando las aplicaciones de publicación y los suscriptores continúan operando, los mensajes que están en cola se colocarán en la cola de mensajes no entregados y se interrumpirán las suscripciones.

Si no pueden producirse estas situaciones, las publicaciones siempre se entregan por orden.

Nota: No se pueden utilizar mensajes agrupados o segmentados con la publicación/suscripción.

Interceptación de publicaciones

Se puede interceptar una publicación, modificarla y publicarla antes de que llegue a cualquier otro suscriptor.

Puede que desee interceptar una publicación antes de que llegue a un suscriptor a fin de realizar una de las acciones siguientes:

- Adjuntar información adicional al mensaje.
- Bloquear el mensaje.
- Transformar el mensaje.

Se puede realizar la misma operación en cada mensaje, o variar la operación en función de la suscripción, del mensaje o de la cabecera del mensaje.

Referencia relacionada

MQ_PUBLISH_EXIT - Salida de publicación

Niveles de suscripción

Establezca el nivel de una suscripción para interceptar una publicación antes de que llegue a sus suscriptores finales. Un suscriptor de interceptación se suscribe en un nivel de suscripción más alto y vuelve a publicar en un nivel de publicación más bajo. Cree una cadena de suscriptores de interceptación para procesar mensajes de una publicación antes de que su entrega a los suscriptores finales.

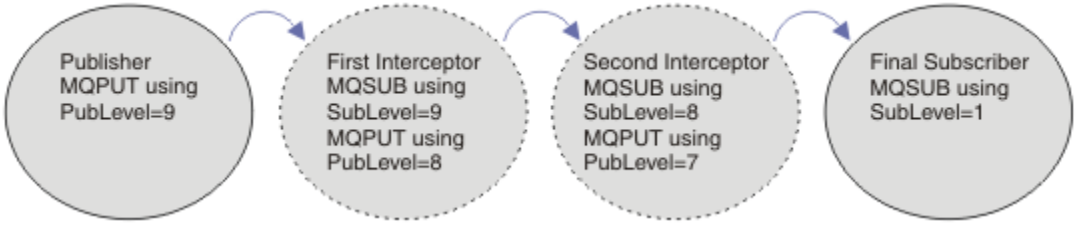


Figura 86. Secuencia de suscriptores de interceptación

Para interceptar una publicación utilice el atributo SubLevel de **MQSD**. Una vez interceptado un mensaje, se puede transformar y, a continuación, volver a publicar en un nivel de publicación más bajo cambiando el atributo PubLevel de **MQPMO**. A continuación, el final llega a los suscriptores finales o lo vuelve a interceptar un suscriptor intermedio en un nivel de suscripción inferior.

Normalmente, el suscriptor de interceptación transforma un mensaje antes de volver a publicarlo. Una secuencia de suscriptores de interceptación crea un flujo de mensajes. De forma alternativa, es posible que no vuelva a publicar la publicación interceptada: Los suscriptores de los niveles de suscripción inferiores no recibirán el mensaje.

Asegúrese de que el receptor recibe las publicaciones antes que cualquier otro suscriptor. Establezca el nivel de suscripción del interceptor en un valor más alto que el de otros suscriptores. De forma predeterminada, los suscriptores tienen un SubLevel de 1. El valor más alto es 9. Una publicación debe empezar con un PubLevel al menos tan alto como el SubLevelmás alto. Inicialmente, publique con el valor predeterminado de PubLevel de 9.

- Si tiene un suscriptor de interceptación en un tema establezca el valor de SubLevel en 9.
- En el caso de varias aplicaciones de interceptación para un tema, establezca un valor de SubLevel más bajo para cada suscriptor de interceptación sucesivo.
- Puede implementar un máximo de 8 aplicaciones de interceptación, con niveles de suscripción de 9 hasta 2 inclusive. El destinatario final del mensaje tiene un valor de SubLevel de 1.

El interceptor con el nivel de suscripción más alto que sea igual o menor que el valor de PubLevel de la publicación, recibe la primera publicación. Configure solo un suscriptor de interceptación para un tema en un nivel de suscripción concreto. Si tiene varios suscriptores en un nivel de suscripción concreto se enviarán varias copias de la publicación al conjunto final de aplicaciones suscriptoras.

Un suscriptor con un valor de SubLevel de 0 se utiliza como suscriptor global. Recibe la publicación si ningún suscriptor final recibe el mensaje. Se puede utilizar un suscriptor con un valor de SubLevel de 0 para supervisar las publicaciones que no recibe ningún otro suscriptor.

Programación de un suscriptor de interceptación

Utilice las opciones de suscripción que se describen en Tabla 125 en la página 862.

Tabla 125. Opciones de suscripción para suscriptores de interceptación

Opción de suscripción	**Notas**
MQSO_SET_CORREL_ID y SubCorrelId establecidas en MQCI_NONE	Mantenga el CorrelId de la publicación interceptada en el mismo valor que la publicación original. **Nota:** No puede pasar el identificador de correlación de una publicación en una jerarquía. El campo lo utiliza el gestor de colas.
PubPriority establecido en MQPRI_PRIORITY_AS_PUBLISHED	Mantener la prioridad de la publicación interceptada en el mismo valor que la publicación original.

Las opciones de la Tabla 125 en la página 862 las deben utilizar todos los suscriptores de interceptación. El resultado es que el identificador de correlación y la prioridad del mensaje no se modifican al establecer el publicador original.

Cuando el suscriptor de interceptación ha procesado la publicación, vuelve a publicar el mensaje para el mismo tema en un valor de `PubLevel` de un número menos que el valor de `SubLevel` de su propia suscripción. Si el suscriptor de interceptación se establece en un valor de `SubLevel` de 9, vuelve a publicar el mensaje con un valor de `PubLevel` de 8.

Para volver a publicar correctamente el mensaje, son necesarias varias partes de la información de la publicación original. Reutilice el mismo **MQMD** que en el mensaje original y establezca `MQPMO_PASS_ALL_CONTEXT` para asegurarse de que toda la información de **MQMD** se pasa al siguiente suscriptor. Copie los valores de las propiedades del mensaje que se muestran en la Tabla 126 en la página 863 en los campos correspondientes del mensaje que se ha vuelto a publicar. El suscriptor de interceptación puede cambiar estos valores. Utilice el operador OR para añadir valores adicionales a **MQPMO**. El campo `Options`, para combinar las opciones de transferencia de mensajes.

Debe abrir la cola de publicación de forma explícita, en lugar de utilizar una cola de publicación gestionada. No puede establecer `MQSO_SET_CORREL_ID` para una cola gestionada. Tampoco puede establecer `MQOO_SAVE_ALL_CONTEXT` en un cola gestionada. Consulte los fragmentos de código que se muestran en la sección "Ejemplos" en la página 863.

Tabla 126. Valores de MQPUT para mensajes republicados

Volver a publicar mensaje utilizando MQPUT	Información del mensaje de publicación
MQOD. `ObjectString`	Propiedad de mensaje `MQTopicString`
MQPMO. `Options`	Propiedad de mensaje `MQPubOptions`

El suscriptor final puede optar por establecer sus opciones de suscripción de forma diferente. Por ejemplo, puede establecer la prioridad de la publicación de forma explícita, en lugar de establecerla en `MQPRI_PRIORITY_AS_PUBLISHED`. Los valores de un suscriptor final solo afectan a la publicación del suscriptor de interceptación final de la cadena.

Publicaciones retenidas

Una publicación retenida se debe conservar después de que haya interceptada, copiando sus opciones de colocación de mensaje originales en el mensaje republicado.

La opción `MQPMO_RETAIN` la establece la aplicación de publicación. Cada suscriptor de interceptación debe transferir el `MQPubOptions` a las opciones de transferencia de mensajes del mensaje que se ha vuelto a publicar, tal como se muestra en la Tabla 126 en la página 863. Copiar las opciones de colocación de mensajes conserva las opciones que ha establecido el publicador original, incluido si se ha de retener la publicación.

Cuando una publicación finaliza su paso descendente por la cadena de suscriptores de interceptación y se entrega a los suscriptores finales, se retiene finalmente. Los nueves suscriptores, con un `SubLevel` de 1 que solicitan la publicación retenida, la reciben si ninguna interceptación adicional. La publicación retenida no se envía a los suscriptores con un `SubLevel` mayor que 1. Por lo tanto, la publicación retenida no resulta modificada por la cadena de suscriptores de interceptación por segunda vez.

Ejemplos

Los ejemplos son fragmentos de código que se pueden combinar para crear una suscriptor de interceptación. El código se ha abreviado, y no muestra la calidad de producción.

Las directivas de preprocesador de la Figura 87 en la página 864 definen las dos propiedades que se han de extraer de los mensajes de publicación que requiere la llamada MQI `MQINQMP`.

```
#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
```

Figura 87. Directivas de preprocesador

La Figura 88 en la página 864 lista las declaraciones que se utilizan en los fragmentos de código. Salvo los términos resaltados, las declaraciones son un estándar para una aplicación IBM MQ.

Las opciones Put y Get resaltadas se inicializan para pasar todo el contexto. Las opciones MQTOPICSTRING y MQPUBOPTIONS resaltadas son inicializadores MQCHARV para nombres de propiedades definidos en las directivas de preprocesador. Los nombres se pasan a MQINQMP.

```
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;
```

Figura 88. Declaraciones

En la Figura 89 en la página 865 se muestran las inicializaciones que no se realizan fácilmente en las declaraciones. Los valores resaltados requieren una descripción.

SYSTEM.NDURABLE.MODEL.QUEUE
: En este ejemplo, en lugar de utilizar MQSUB para abrir una suscripción gestionada no duradera, se utiliza la cola del modelo, SYSTEM.NDURABLE.MODEL.QUEUE, para crear una cola dinámica

temporal. Su descriptor se pasa a MQSUB. Al abrir directamente la cola puede guardar el contexto del mensaje completo y establecer la opción de suscripción MQSO_SET_CORREL_ID.

MQGMO_CURRENT_VERSION
Es importante utilizar la versión actual de la mayor parte de las estructuras IBM MQ. Los campos, tales como gmo.MsgHandle, solo están disponibles en la versión más reciente de las estructuras de control.

MQGMO_PROPERTIES_IN_HANDLE
La serie de tema y las opciones de colocación de mensajes establecidas en la publicación original las ha de recuperar el suscriptor de interceptación utilizando las propiedades del mensaje. Una alternativa puede ser leer directamente la estructura **MQRFH2** en el mensaje.

MQSO_SET_CORREL_ID
Utilice MQSO_SET_CORREL_ID junto con

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

Estas opciones hacen que se pase el identificador de correlación. El identificador de correlación establecido por el publicador original se coloca en el campo del identificador de correlación de la publicación que recibe el suscriptor de interceptación. Cada suscriptor de interceptación se pasa en el mismo identificador de correlación. A continuación, el suscriptor final tiene la opción de recibir el mismo identificador de correlación.

Nota: Si la publicación se pasa a través de una jerarquía de publicación/suscripción, no se retiene nunca el identificador de correlación.

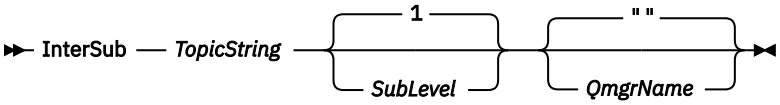
MQPRI_PRIORITY_AS_PUBLISHED
La publicación se coloca en la cola de publicación con la misma prioridad de mensaje con la que se ha publicado.

```
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;
```

Figura 89. Inicializaciones

La Figura 90 en la página 866 muestra el fragmento de código para leer parámetros de línea de mandatos, completar la inicialización y crear la suscripción interceptora.

Ejecute el programa con el mandato,



Para que el manejo de errores cree el menor número de interrupciones posible, el código de razón de cada llamada MQI se almacena en un elemento de matriz diferente. Después de cada llamada se prueba

el código de terminación, y si el valor es MQCC_FAIL, el control emite el bloque de código `do { } while(0)`.

Las dos líneas de código a tener en cuenta son,

pmo.PubLevel = sd.SubLevel - 1;
Establece el nivel de publicación del mensaje republicado en un número menos que el nivel de suscripción del suscriptor de interceptación.

gmo.MsgHandle = Hmsg;
Proporciona un descriptor de mensaje para que MQGET devuelva las propiedades del mensaje.

```
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;
```

Figura 90. Preparación de la interceptación de publicaciones

El fragmento de código principal, Figura 91 en la página 867, obtiene los mensajes de la cola de publicación. Consulta las propiedades del mensaje y vuelve a publicar los mensajes utilizando la serie de tema y el **MQPMO** original. Las propiedades de `option` de la publicación.

En este ejemplo, no se realiza ninguna transformación en la publicación. La serie de tema de la publicación que se ha vuelto a publicar siempre coincide con la serie de tema del suscriptor de interceptación suscrito. Si el suscriptor de interceptación es el responsable de interceptar varias suscripciones enviadas a la misma cola de publicación, es posible que sea necesaria consultar la serie de tema para diferenciar las publicaciones que coinciden con diferente suscripciones.

Las llamadas a MQINQMP están resaltadas. La serie de tema y las propiedades de las opciones de transferencia de mensajes de publicación se escriben directamente en las estructuras de control de salida. La única razón para alterar el campo de longitud MQCHARV de `putOD.ObjectString` de una longitud explícita a una serie terminada en nulos es utilizar `printf` para generar la serie.

```
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;
}
```

Figura 91. Interceptar una publicación y volver a publicarla

El fragmento de código final se muestra en la Figura 92 en la página 867.

```
    } 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);
}
```

Figura 92. Terminación

Interceptación de publicaciones y publicación/suscripción distribuida
Siga un patrón simple cuando despliegue suscriptores de interceptación o salidas de publicación en una topología de publicación/suscripción distribuida. Despliegue los suscriptores de interceptación en los mismos gestores de colas que los publicadores, y las salidas de publicación en los mismos gestores de colas que los suscriptores finales.

Figura 93 en la página 868 muestra dos gestores de colas conectados en un clúster de publicación/suscripción. Un publicador crea una publicación en un tema de clúster a nivel de publicación 9. Las flechas numeradas muestran la secuencia de pasos realizados por la publicación a medida que fluye a los suscriptores al tema de clúster. La publicación es interceptada por el suscriptor con `Subnivel` 9 y se vuelve a publicar con `Publevel` 8. Es interceptado de nuevo por un suscriptor en `Subnivel` 8. El suscriptor vuelve a publicar en `Publevel` 7. El suscriptor proxy proporcionado por el gestor de colas reenvía la publicación al gestor de colas B, donde se ha desplegado una salida de publicación además de un suscriptor final. La salida de publicación procesa la publicación antes de que finalmente la reciba el suscriptor final en `Subnivel` 1. Los suscriptores de interceptación y la salida de publicación se muestran con esquemas rotos.

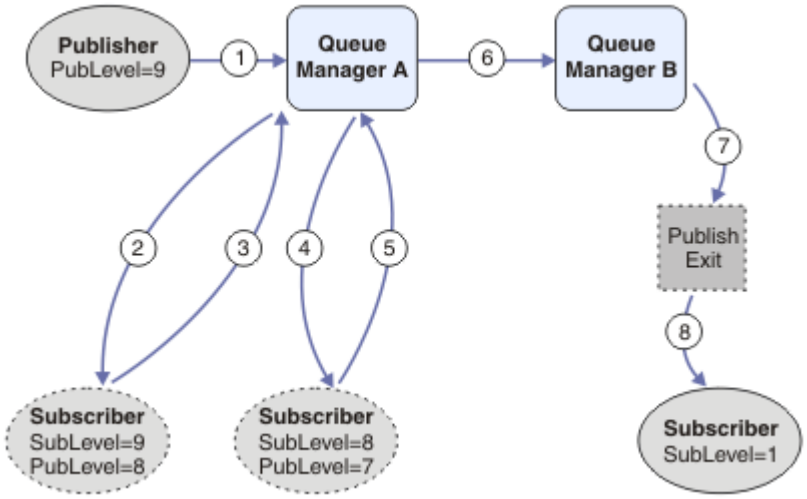


Figura 93. Salida de interceptación y publicación en un clúster

El objetivo del patrón simple es que cada suscriptor que reciba una publicación la reciba idéntica. La publicación pasa por la misma secuencia de transformaciones independientemente del lugar en el que esté conectado el suscriptor. Probablemente desee evitar que la secuencia de transformaciones varíe en función de dónde estén conectados los publicadores o los suscriptores finales. Una excepción razonable sería adaptar la publicación finalmente entregada a cada suscriptor individual. Utilice la salida de publicación para personalizar la publicación en función de la cola a la que finalmente se entregue.

Hay que pensar cuidadosamente dónde desplegar los suscriptores de interceptación y las salidas de publicación en una topología de publicación/suscripción. El patrón sencillo despliega suscriptores de interceptación en el mismo gestor de colas que los publicadores y las salidas de publicación en los mismos gestores de colas que los suscriptores finales.

Antipatrón

Figura 94 en la página 869 muestra cómo las cosas pueden salir mal si no se sigue un patrón simple. Para complicar el despliegue, se añade un suscriptor final al gestor de colas A y se añaden dos suscriptores de interceptación adicionales al gestor de colas B.

La publicación se reenvía al gestor de colas B en `PubLevel` 7, donde es interceptado por un suscriptor en `SubLevel` 5 antes de ser consumido por el suscriptor final en `SubLevel` 1. La salida de publicación intercepta la publicación antes de que se pase al consumidor de interceptación y al consumidor final en el gestor de colas B. La publicación alcanza el suscriptor final en el gestor de colas A sin que la salida de publicación lo procese.

En una topología de publicación/suscripción, los suscriptores de proxy se suscriben en `Sublevel` 1 y pasan el `PubLevel` establecido por el último suscriptor de interceptación. En Figura 94 en la página 869, el resultado es que la publicación no es interceptada por el suscriptor utilizando `Sublevel` 9 en el gestor de colas B.

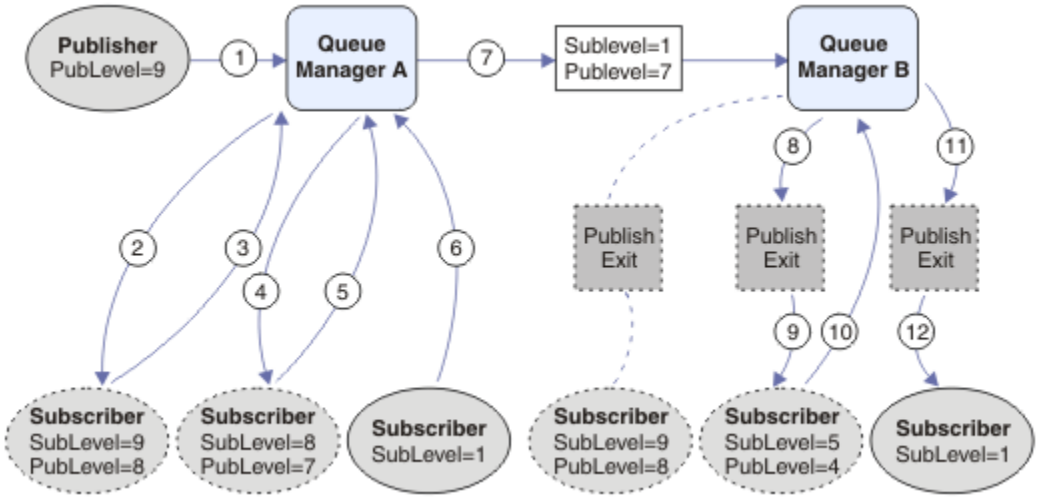


Figura 94. Despliegue complejo de suscriptores de interceptación

Opciones de publicación

Existen varias opciones que controlan la forma en que se publican los mensajes.

Retención de la información de respuesta de los suscriptores

Si no desea que los suscriptores puedan responder a las publicaciones que reciben, es posible retener la información de los campos ReplyToQ y ReplyToQmgr del MQMD utilizando la opción de colocación de mensaje MQPMO_SUPPRESS_REPLYTO. Si se utiliza esta opción, el gestor de colas elimina dicha información del MQMD cuando recibe la publicación antes de reenviarla a los suscriptores.

Esta opción no se puede usar junto con una opción de informe que requiera un ReplyToQ; si se intenta esto, la llamada fallará con MQRC_MISSING_REPLY_TO_Q.

Nivel de publicación

El uso de los niveles de publicación es una forma de controlar los suscriptores que reciben una publicación. El nivel de publicación indica el nivel de suscripción al que va dirigida la suscripción. Únicamente las suscripciones cuyo nivel de suscripción más alto sea menor o igual que el nivel de la publicación, recibirán dicha publicación. Este valor tiene que estar en el rango de cero a nueve; cero es el nivel de publicación más bajo. El valor inicial de este campo es 9. Uno de los usos de los niveles de publicación y suscripción es interceptar publicaciones.

Comprobación de la entrega de una publicación a los suscriptores

Para comprobar si una publicación no se ha entregado a los suscriptores, utilice la opción de colocación de mensaje MQPMO_WARN_IF_IF_NO_SUBS_MATCHED con la llamada MQPUT. Si la operación de colocación devuelve un código de terminación de MQCC_WARNING y un código de razón MQRC_NO_SUBS_MATCHED, la publicación no se ha entregado a ninguna suscripción. Si se especifica la opción MQPMO_RETAIN en la operación de colocación, el mensaje se retiene y se entregará a cualquier suscripción coincidente que se defina posteriormente. En un sistema de publicación/suscripción distribuido, el código de razón MQRC_NO_SUBS_MATCHED solo se devuelve si no hay suscripciones de proxy registradas para el tema en el gestor de colas.

Opciones de suscripción

Hay varias opciones disponibles que controlan la forma en que se manejan las suscripciones de mensajes.

Persistencia de los mensajes

Los gestores de colas mantienen la persistencia de las publicaciones que reenvían a los suscriptores según lo establecido por el publicador. El publicador establece la persistencia en una de las siguientes opciones:

0
No persistente

1
Persistente

2
Persistencia como definición de cola/tema

Para la publicación/suscripción, el publicador resuelve el objeto de tema y **topicString** en un objeto de tema resuelto. Si el publicador especifica Persistencia como definición de cola/tema, se establece la persistencia predeterminada del objeto de tema resuelto para la publicación.

Publicaciones retenidas

Para controlar cuándo se reciben publicaciones retenidas, los suscriptores pueden utilizar dos opciones de suscripción:

Publicar solo a petición, MQSO_PUBLICATIONS_ON_REQUEST

Si desea que un suscriptor tenga el control cuando recibe publicaciones, puede utilizar la opción de suscripción MQSO_PUBLICATIONS_ON_REQUEST. Un suscriptor puede controlar cuándo recibe publicaciones utilizando la llamada MQSUBRQ (especificando el manejador Hsub que se ha devuelto de la llamada MQSUB original) para solicitar que se envíe una publicación retenida de un tema. Los suscriptores que utilizan la opción de suscripción MQSO_PUBLICATIONS_ON_REQUEST no reciben ninguna publicación no retenida.

Si especifica MQSO_PUBLICATIONS_ON_REQUEST, debe utilizar MQSUBRQ para recuperar cualquier publicación. Si no utiliza MQSO_PUBLICATIONS_ON_REQUEST, recibirá los mensajes a medida que se publiquen.

Si un suscriptor utiliza la llamada MQSUBRQ y utiliza comodines en el tema de la suscripción, la suscripción puede coincidir con varios temas o nodos en un árbol de temas; todos los que tengan mensajes retenidos (si existen) se enviarán al suscriptor.

Esta opción puede ser especialmente útil cuando se utiliza con suscripciones duraderas porque un gestor de colas continuará enviando publicaciones a un suscriptor si se suscribió como duradera, aunque esa aplicación de suscriptor no esté ejecutándose. Esto puede provocar una acumulación de mensajes en la cola de suscriptores. Esta acumulación puede evitarse si el suscriptor se registra utilizando la opción MQSO_PUBLICATIONS_ON_REQUEST. Como alternativa, puede utilizar suscripciones no duraderas si son adecuadas para la aplicación, para evitar una acumulación de mensajes no deseados.

Si una suscripción es duradera y un publicador utiliza publicaciones retenidas, la aplicación de suscriptor puede utilizar la llamada MQSUBRQ para renovar su información de estado después de un reinicio. A continuación, el suscriptor deberá renovar su estado periódicamente utilizando la llamada MQSUBRQ.

No se enviarán publicaciones como consecuencia de que la llamada MQSUB utilice esta opción. Una suscripción duradera que se ha reanudado después de la desconexión utilizará la opción MQSO_PUBLICATIONS_ON_REQUEST si la suscripción original se ha configurado para utilizar esta opción.

Solo publicaciones nuevas, MQSO_NEW_PUBLICATIONS_ONLY

Si existe una publicación retenida sobre un tema, todos los suscriptores que realicen una suscripción después de que se haya realizado la publicación recibirán una copia de dicha publicación. Si un suscriptor no desea recibir ninguna de las publicaciones que se realizaron anteriormente

a la suscripción que se realiza ahora, el suscriptor puede utilizar la opción de suscripción MQSO_NEW_PUBLICATIONS_ONLY.

Agrupación de suscripciones

Se recomienda agrupar las suscripciones si ha configurado una cola para recibir publicaciones y tiene una serie de suscripciones que se solapan que suministran publicaciones a la misma cola. Esta situación es similar al ejemplo de Solapamiento de suscripciones.

Para evitar recibir publicaciones duplicadas, establezca la opción MQSO_GROUP_SUB cuando se suscriba a un tema. El resultado es que cuando haya más de una suscripción en el grupo que coincida con el tema de una publicación, una sola suscripción será responsable de colocar la publicación en la cola. Las otras suscripciones que coincidan con el tema de la publicación se ignorarán.

La suscripción responsable de colocar la publicación en la cola se elige sobre la base de que tiene la serie de tema coincidente más larga, antes de encontrar ningún comodín. Puede considerarse como la suscripción con mayor grado de coincidencia. Sus propiedades se propagan a la publicación, incluyendo si tiene o no la propiedad MQSO_NOT_OWN_PUBS. Si lo hace, no se entrega ninguna publicación a la cola, aunque es posible que otras suscripciones coincidentes no tengan la propiedad MQSO_NOT_OWN_PUBS.

No puede colocar todas las suscripciones en un solo grupo para eliminar publicaciones duplicadas. Las suscripciones agrupadas deben cumplir estas condiciones:

1. Ninguna de las suscripciones están gestionadas.
2. Un grupo de suscripciones entregan publicaciones a la misma cola.
3. Cada suscripción debe estar en el mismo nivel de suscripción.
4. El mensaje de publicación para cada suscripción del grupo tiene el mismo identificador de correlación.

   Para asegurar que cada suscripción tenga como resultado un mensaje de publicación con el mismo identificador de correlación, establezca MQSO_SET_CORREL_ID para crear su propio identificador de correlación en la publicación, y establezca el mismo valor en el campo **SubCorrelId** en cada suscripción. No establezca **SubCorrelId** en el valor MQCI_NONE.

Consulte ../refdev/q100080_.dita#q100080_/mqso_group_sub para obtener más información.

Consulta y establecimiento de atributos de objeto

Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

Afectan a la forma en que un gestor de colas procesa un objeto. Los atributos de cada tipo de objeto de IBM MQ se describen detalladamente en Atributos de objetos.

Algunos atributos se establecen cuando se define el objeto y solo se pueden cambiar utilizando los mandatos de IBM MQ; un ejemplo de este atributo es la prioridad predeterminada de los mensajes colocados en una cola. Otros atributos se ven afectados por la operación del gestor de colas y pueden cambiar en el tiempo; un ejemplo sería la profundidad actual de una cola.

Puede consultar los valores actuales de la mayoría de los atributos utilizando la llamada MQINQ. MQI también proporciona una llamada MQSET con la que puede cambiar algunos atributos de cola. No puede utilizar las llamadas MQI para cambiar los atributos de cualquier otro tipo de objeto. En su lugar, debe utilizar uno de los recursos siguientes:

- ALW El recurso MQSC, que se describe en los mandatos MQSC.
- IBM i Los mandatos CL de CHGMQMx, que se describen en Referencia de mandatos CL de IBM i o en el recurso MQSC.
- z/OS Los mandato de operador ALTER o los mandatos DEFINE con la opción REPLACE, que se describen en Mandatos MQSC.

Nota: Los nombres de los atributos de los objetos se muestran en esta documentación con el formato que los utiliza con las llamadas MQINQ y MQSET. Cuando utiliza mandatos de IBM MQ para definir, alterar o

visualizar los atributos, debe identificar los atributos utilizando las palabras clave que se muestran en las descripciones de los mandatos de los enlaces de tema.

Las llamadas MQINQ y MQSET utilizan matrices de selectores para identificar los atributos que desea consultar o establecer. Hay un selector para cada atributo con el que puede trabajar. El nombre de selector tiene un prefijo, que viene determinado por la naturaleza del atributo:

Tabla 127. Prefijos de los nombres de selector

Prefijo	**Descripción**
MQCA_	Estos selectores hacen referencia a atributos que contienen datos de tipo carácter (por ejemplo, el nombre de una cola).
MQIA_	Estos selectores hacen referencia a atributos que contienen valores numéricos (por ejemplo, *CurrentQueueDepth*, el número de mensajes en una cola) o un valor constante (por ejemplo, *SyncPoint*, si el gestor de colas da soporte a los puntos de sincronización).

Para poder utilizar las llamadas MQINQ o MQSET, la aplicación debe estar conectada al gestor de colas, y debe utilizar la llamada MQOPEN para abrir el objeto para establecer o consultar los atributos. Estas operaciones se describen en "Conexión y desconexión de un gestor de colas" en la página 754 y "Apertura y cierre de objetos" en la página 761.

Utilice los enlaces siguientes para obtener más información sobre cómo consultar y establecer los atributos de objeto:

- "Consulta de los atributos de un objeto" en la página 873
- "Algunos casos en los que falla la llamada MQINQ" en la página 873
- "Cómo establecer los atributos de cola" en la página 874

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Apertura y cierre de objetos" en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

"Colocación de mensajes en una cola" en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911

Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Consulta de los atributos de un objeto

Utilice la llamada MQINQ para consultar sobre los atributos de cualquier tipo de IBM MQ.

Como entrada para esta llamada, debe proporcionar:

- Un descriptor de conexión.
- Un descriptor de objeto.
- El número de selectores.
- Una matriz de selectores de atributos, cada selector con el formato MQCA_ * o MQIA_ *. Cada selector representa un atributo con un valor sobre el que desea realizar una consulta, y cada selector debe ser válido para el tipo de objeto que representa el descriptor de objeto. Los selectores se pueden especificar en cualquier orden.
- El número de atributos de entero que se consultan. Especifique cero si no va a consultar ningún atributo entero.
- La longitud del almacenamiento intermedio de atributos de caracteres en *CharAttrLength*. Como mínimo, tiene que ser la suma de las longitudes necesarias para alojar todas las cadenas de atributo de carácter. Especifique cero si no va a consultar ningún atributo de carácter.

La salida de MQINQ es:

- Un conjunto de valores de atributo de entero copiados en el vector. El número de valores viene determinado por *IntAttrCount*. Si *IntAttrCount* o *SelectorCount* es cero, este parámetro no se utiliza.
- El búfer donde se devuelven los atributos de carácter. La longitud del búfer se proporciona en el parámetro **CharAttrLength**. Si *CharAttrLength* o *SelectorCount* es cero, este parámetro no se utiliza.
- Un código de terminación. Si el código de terminación indica un aviso, esto significa que la llamada solo se ha completado parcialmente. En tal caso, examine el código de razón.
- Un código de razón. Hay tres situaciones de terminación parcial:
  - El selector no se aplica al tipo de cola.
  - No hay espacio suficiente para los atributos de entero.
  - No hay espacio suficiente para los atributos de carácter.

  Si se producen más de una de estas situaciones, se devuelve la primera que aplique.

Si se abre una cola para salida o consulta y resuelve a una cola de clúster no local, solo se podrán consultar el nombre, el tipo y los atributos comunes de de dicha cola. Los valores de los atributos comunes son los de la cola seleccionada si se ha usado MQOO_BIND_ON_OPEN. Los valores serán los de una cola cualquiera de las posibles del clúster si se utilizan MQOO_BIND_NOT_FIXED o MQOO_BIND_ON_GROUP o MQOO_BIND_AS_Q_DEF y el atributo de cola **DefBind** es MQBND_BIND_NOT_FIXED. Consulte "La llamada MQOPEN y los clústeres" en la página 907 y MQOPEN para obtener información adicional.

Nota: Los valores devueltos por la llamada son una instantánea de los atributos seleccionados. Estos pueden cambiar antes de que el programa actúe sobre los valores devueltos.

En MQINQ hay una descripción de la llamada MQINQ.

Algunos casos en los que falla la llamada MQINQ

Si abre un alias para consultar sobre sus atributos, se le devuelven los atributos de la cola de alias (el objeto IBM MQ utilizado para acceder a otra cola), no los de la cola base.

Sin embargo, la definición de la cola base a la que resuelve el alias también la abre el gestor de colas, y si otro programa cambia el uso de la cola base en el intervalo que transcurre entre las llamadas MQOPEN y MQINQ, la llamada MQINQ fallará y devolverá el código de razón MQRC_OBJECT_CHANGED. La llamada también falla si se modifican los atributos del objeto de cola alias.

De forma similar, cuando abre una cola remota para consultar sus atributos, solo se devuelven los atributos de la definición local de la cola remota.

Si se especifican uno o más selectores que no son válidos para el tipo de atributos de cola que se están consultando, la llamada MQINQ completa con un aviso y establece la salida como se indica a continuación:

- Para atributos enteros, los elementos correspondientes de *IntAttrs* se establecen en MQIAV_NOT_APPLICABLE.
- Para los atributos de caracteres, las partes correspondientes de la serie *CharAttrs* se establecen en asteriscos.

Si se especifican uno o más selectores que no son válidos para el tipo de atributos de objeto consultados, la llamada MQINQ fallará y devolverá el código de razón MQRC_SELECTOR_ERROR.

No puede invocar MQINQ para consultar una cola modelo; utilice el recurso MQSC o los comandos disponibles en la plataforma.

Cómo establecer los atributos de cola

Utilice esta información para aprender cómo establecer los atributos de cola utilizando la llamada MQSET.

Utilizando la llamada MQSET, sólo puede establecer los atributos de cola siguientes:

- *InhibitGet* (pero no para colas remotas)
- *DistList* (no en z/OS)
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

La llamada MQSET tiene los mismos parámetros que la llamada MQINQ. No obstante, para MQSET, todos los parámetros, excepto el código de terminación y el código de razón, son parámetros de entrada. No hay situaciones de finalización parcial.

Nota: No se puede utilizar MQI para establecer los atributos de los objetos de IBM MQ que no sean colas definidas localmente.

Para obtener más información sobre la llamada MQSET, consulte MQSET.

Confirmación y restitución de unidades de trabajo

Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

En este tema se utilizan los términos siguientes:

- Confirmar
- Restituir
- Coordinación de puntos de sincronización
- Punto de sincronismo
- Unidad de trabajo
- confirmación en una sola fase

- Confirmación en dos fases

Si conoce estos términos del proceso de transacciones, puede ir directamente a la sección "Consideraciones acerca del punto de sincronismo en aplicaciones IBM MQ" en la página 876.

Confirmar y restituir
Cuando un programa coloca un mensaje en una cola dentro de una unidad de trabajo, ese mensaje pasa a ser visible para los demás programas solo cuando el programa confirma la unidad de trabajo. Para confirmar una unidad de trabajo, todas las actualizaciones deben realizarse satisfactoriamente para mantener la integridad de los datos. Si el programa detecta un error y determina que la operación de transferir no es permanente, puede restituir la unidad de trabajo. Cuando un programa realiza una restitución, IBM MQ restaura la cola eliminando los mensajes que la unidad de trabajo colocó en la cola. La forma en que el programa realiza las operaciones de confirmación y restitución depende del entorno en el que se ejecuta el programa.

De forma similar, cuando un programa obtiene un mensaje de una cola dentro de una unidad de trabajo, ese mensaje permanece en la cola hasta que el programa confirma la unidad de trabajo, pero el mensaje no puede ser recuperado por otros programas. El mensaje se suprime de forma permanente de la cola cuando el programa confirma la unidad de trabajo. Si el programa restituye la unidad de trabajo, IBM MQ restaura la cola haciendo que los mensajes puedan ser recuperados por otros programas.

Coordinación de puntos de sincronización, punto de sincronización, unidad de trabajo
La *Coordinación de puntos de sincronización* es el proceso por el que las unidades de trabajo se confirman o restituyen preservando la integridad de los datos.

La decisión de confirmar o restituir los cambios se toma, en el caso más sencillo, al final de una transacción. Pero puede ser más útil para una aplicación sincronizar los cambios de datos en otros puntos lógicos dentro de una transacción. Estos puntos lógicos se denominan *puntos de sincronización* (o *puntos de sincronización*) y el periodo de proceso de un conjunto de actualizaciones entre dos puntos de sincronización se denomina *unidad de trabajo*. Algunas llamadas MQGET y MQPUT pueden formar parte de una sola unidad de trabajo.

El número máximo de mensajes dentro de una unidad de trabajo puede controlarse mediante el atributo MAXUMSGS del mandato ALTER QMGR .

confirmación en una sola fase
Un proceso de *confirmación en una sola fase* es un proceso en el que un programa puede confirmar actualizaciones realizadas en una cola sin coordinar esos cambios con otros gestores de recursos.

Confirmación en dos fases
Un proceso de *confirmación en dos fases* es un proceso en el que las actualizaciones que un programa ha realizado en colas de IBM MQ se pueden coordinar con actualizaciones realizadas en otros recursos (por ejemplo, bases de datos bajo el control de Db2). En un proceso de esta clase, las actualizaciones hechas en todos los recursos se confirman o restituyen juntas.

Para ayudar a gestionar las unidades de trabajo, IBM MQ proporciona el atributo **BackoutCount**. Este atributo se incrementa cada vez que se restituye un mensaje dentro de una unidad de trabajo. Si el mensaje provoca repetidamente que la unidad de trabajo termine de forma anómala, el valor de *BackoutCount* finalmente es mayor que valor de *BackoutThreshold*. Este último valor se establece cuando se define la cola. En esta situación, la aplicación puede eliminar el mensaje de la unidad de trabajo y colocarlo en otra cola, tal como se define en *BackoutRequeueQName*. Cuando se traslada el mensaje, se puede confirmar la unidad de trabajo.

Utilice los enlaces siguientes para obtener más información sobre la confirmación y restitución de unidades de trabajo:

- "Consideraciones acerca del punto de sincronismo en aplicaciones IBM MQ" en la página 876
- z/OS "Puntos de sincronismo en aplicaciones IBM MQ for z/OS" en la página 877
- IBM i "Puntos de sincronización en CICS para las aplicaciones de IBM i" en la página 880
- "Puntos de sincronización en IBM MQ for Multiplatforms" en la página 880

- IBM i “Interfaces para el gestor de puntos de sincronismo externo de IBM i” en la página 885

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

“IMS y las aplicaciones puente IMS en IBM MQ for z/OS” en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Consideraciones acerca del punto de sincronismo en aplicaciones IBM MQ

Utilice esta información para obtener información acerca del uso de puntos de sincronismo en aplicaciones IBM MQ.

Los entornos siguientes dan soporte a la confirmación de dos fases:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server para z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS Proceso por lotes z/OS con RRS
- Otros coordinadores externos utilizando la interfaz X/Open XA

Los entornos siguientes dan soporte a la confirmación de una fase:

- Multi IBM MQ for Multiplatforms
- z/OS Lote z/OS

Para obtener más información sobre las interfaces externas consulte la sección “Interfaces para gestores de puntos de sincronismo externos en Multiplatforms” en la página 883 y el documento de XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicado por The Open Group. Los

gestores de transacciones, tales como CICS, IMS, Encina y Tuxedo, pueden participar en la confirmación de dos fases, coordinados con otros recursos recuperables. Esto significa que las funciones de puesta en cola que proporciona IBM MQ se pueden trasladar al ámbito de una unidad de trabajo gestionada por el gestor de transacciones.

Los ejemplos que se incluyen con IBM MQ muestran la coordinación de IBM MQ con bases de datos compatibles con XA. Para obtener más información sobre estos ejemplos, consulte "Utilización de programas procedimentales de ejemplo de IBM MQ" en la página 1085.

En su aplicación IBM MQ, puede especificar en cada llamada put y get si desea que la llamada esté bajo control de punto de sincronismo. Para una operación put funcione bajo el control de punto de sincronismo, utilice el valor MQPMO_SYNCPOINT del campo *Options* de la estructura MQPMO cuando invoque MQPUT. Para una operación get, utilice el valor MQGMO_SYNCPOINT en el campo *Options* de la estructura MQGMO. Si no selecciona de forma explícita una opción, la acción predeterminada depende de la plataforma:

- Multi El valor predeterminado del control de punto de sincronismo es NO.
- z/OS El valor predeterminado del control de punto de sincronismo es YES.

Cuando se emite una llamada MQPUT1 con MQPMO_SYNCPOINT, el comportamiento predeterminado cambia, de modo que la operación put se completa de forma asíncrona. Esto puede crear cambio de comportamiento en algunas aplicaciones que se basan en la devolución de determinados campos de las estructuras MQOD y MQMD y que ahora contienen valores no definidos. Una aplicación puede especificar MQPMO_SYNC_RESPONSE para asegurarse de que la operación put se realiza de forma sincronizada y que se completen todos los valores de los campos adecuados.

Cuando su aplicación recibe un código de razón MQRC_BACKED_OUT en la respuesta a una operación MQPUT o MQGET bajo punto de sincronismo, normalmente la aplicación debe restituir la transacción actual utilizando MQBACK y, a continuación, debe volver a intentar la transacción completa, si resulta adecuado. Si la aplicación recibe MQRC_BACKED_OUT en la respuesta a una llamada MQCMIT o MQDISC, no es necesario que invoque MQBACK.

Cada vez que se restituye una llamada MQGET, se incrementa el campo *BackoutCount* de la estructura MQMD del mensaje afectado. Un valor alto de *BackoutCount* indica que el mensaje se ha restituido de forma repetida. Esto puede indicar que existe un problema con este mensaje que deberá examinar. Consulte la sección BackoutCount para obtener información detallada sobre *BackoutCount*.

Excepto en el proceso por lotes de z/OS con RRS, si un programa emite una llamada MQDISC mientras existen solicitudes sin confirmar, se produce un punto de sincronismo implícito. Si el programa finaliza de forma anómala, se realiza una restitución de forma implícita.

z/OS En z/OS, también se produce un punto de sincronismo implícito si el programa finaliza normalmente sin llamar a MQDISC. El programa habrá finalizado normalmente si el TCB conectado a MQ finaliza con normalidad. Cuando se ejecuta bajo z/OS UNIX System Services y Language Environment (LE), se invoca el manejo de condiciones predeterminado para terminaciones anómalas o señales. Los manejadores de condiciones de LE procesan la condición de error y el TCB finaliza con normalidad. En estas condiciones, MQ confirma la unidad de trabajo. Para obtener más información, consulte Introducción al manejo de condiciones de Language Environment.

z/OS En los programas IBM MQ for z/OS, puede utilizar la opción MQGMO_MARK_SKIP_BACKOUT para especificar que si se genera una restitución, no se debe restituir un mensaje (para evitar un bucle de tipo *error-restitución-MQGET*). Para obtener más información acerca de cómo utilizar esta opción, consulte "Omisión de restitución" en la página 820.

Los cambios en los atributos de la cola, mediante la llamada MQSET o los mandatos, no resultan afectados por la confirmación o restitución de unidades de trabajo.

z/OS *Puntos de sincronismo en aplicaciones IBM MQ for z/OS*

Este tema explica cómo utilizar los puntos de sincronización en el gestor de transacciones (CICS y IMS) y las aplicaciones por lotes.

z/OS *Puntos de sincronización en aplicaciones de CICS Transaction Server for z/OS*

En una aplicación CICS, establezca un punto de sincronización utilizando el mandato EXEC CICS SYNCPOINT.

Para restituir todos los cambios al punto de sincronización anterior, se puede usar el comando EXEC CICS SYNCPOINT ROLLBACK. Para obtener más información, consulte la publicación *CICS Application Programming Reference*.

Si hay otro recurso recuperable implicado en la unidad de trabajo, el gestor de colas (junto con el gestor de puntos de sincronización CICS) participa en un protocolo de confirmación en dos fases; de lo contrario, el gestor de colas realiza un proceso de confirmación en una sola fase.

Si una aplicación CICS emite la llamada MQDISC, no se toma ningún punto de sincronización implícito. Si la aplicación cierra normalmente, las colas abiertas se cierran y se produce una confirmación implícita. Si la aplicación cierra de forma anómala, las colas abiertas se cierran y se produce una restitución implícita.

z/OS *Puntos de sincronismo en aplicaciones IMS*

En una aplicación IMS, establezca un punto de sincronización utilizando llamadas IMS como, por ejemplo, GU (get unique) para IOPCB y CHKP (punto de comprobación).

Para restituir todos los cambios desde el punto de comprobación anterior, puede utilizar la llamada ROLB (retrotraer) de IMS. Para obtener más información, consulte la documentación de IMS .

El gestor de colas (junto con el gestor de puntos de sincronización IMS) participa en un protocolo de confirmación en dos fases, si otros recursos recuperables también están implicados en la unidad de trabajo.

El adaptador IMS cierra todos los descriptores abiertos en un punto de sincronismo, excepto en un entorno por lotes o en un entorno BMP no controlado por mensajes. Esto es debido a que un usuario diferente podría iniciar la siguiente unidad de trabajo y cuando se emiten las llamadas MQCONN, MQCONNX y MQOPEN, se realiza la comprobación de seguridad de IBM MQ, y no cuando se emiten las llamadas MQPUT o MQGET.

No obstante, en un entorno WFI (Wait-for-Input) o en un entorno PWFI (Pseudo Wait-for-Input), IMS no notifica a IBM MQ que cierre los descriptores hasta que llegue el mensaje siguiente ni se devuelve un código de estado QC a la aplicación. Si la aplicación está esperando en la región de IMS y cualquiera de estos manejadores pertenece a las colas desencadenadas, no se llevará a cabo el desencadenamiento porque las colas están abiertas. Por este motivo, hay que hacer un MQCLOSE explícito de las aplicaciones que ejecutan en un entorno de WFI o antes de hacer un GU del IOPCB del siguiente mensaje.

Si una aplicación IMS, ya sea BMP o MPP, emite la llamada MQDISC, se cierran las colas abiertas pero no se toma implícitamente un punto de sincronismo. Si la aplicación cierra normalmente, las colas abiertas se cierran y se produce una confirmación implícita. Si la aplicación cierra de forma anómala, las colas abiertas se cierran y se produce una restitución implícita.

z/OS *Puntos de sincronización en aplicaciones por lotes de z/OS*

Para las aplicaciones por lotes, puede utilizar las llamadas de gestión de punto de sincronización de IBM MQ: MQCMIT y MQBACK. Por motivos de compatibilidad con versiones anteriores, CSQBCMT y CSQBBAK están disponibles como sinónimos.

Nota: Si necesita confirmar o restituir actualizaciones sobre recursos gestionados por distintos gestores de recursos, como IBM MQ y Db2, dentro de una unidad de trabajo individual, puede utilizar RRS. Para obtener más información, consulte “Servicios de gestión de transacciones y gestor de recursos recuperables” en la página 879.

Confirmación de cambios utilizando la llamada MQCMIT

Como entrada, debe proporcionar el descriptor de conexión (*Hconn*) que devuelve la llamada MQCONN o MQCONNX.

La salida de MQCMIT es un código de terminación y un código de razón. La llamada finaliza con un aviso si el punto de sincronización se ha completado pero el gestor de colas ha restituido las operaciones put (colocar) y get (obtener) desde el punto de sincronización anterior.

Una finalización correcta de la llamada MQCMIT indica al gestor de colas que la aplicación ha alcanzado un punto de sincronización y que todas las operaciones put y get realizadas desde el punto de sincronización anterior han pasado a ser permanentes.

No todas las respuestas de error implican que MQCMIT no se ha completado. Por ejemplo, la aplicación puede recibir MQRC_CONNECTION_BROKEN.

Hay una descripción de la llamada MQCMIT en MQCMIT.

Restitución de cambios utilizando la llamada MQBACK

Como entrada, debe proporcionar un descriptor de conexión (*Hconn*). Utilice el descriptor de contexto devuelto por la llamada MQCONN o MQCONNX.

La salida de MQBACK es un código de terminación y un código de razón.

La salida indica al gestor de colas que la aplicación ha alcanzado un punto de sincronización y que todas las operaciones get y put que se hayan realizado desde el último punto de sincronización se han restituido.

Hay una descripción de la llamada MQBACK en MQBACK.

Servicios de gestión de transacciones y gestor de recursos recuperables

Los servicios de gestión de transacciones y gestor de recursos recuperables (RRS) es un recurso de z/OS para proporcionar soporte de punto de sincronización de dos fases en los gestores de recursos participantes.

Una aplicación puede actualizar los recursos recuperables gestionados por diversos gestores de z/OS como IBM MQ y Db2, y luego confirmar o restituir estas actualizaciones como una unidad de trabajo individual. RRS proporciona el registro de estado de unidad de trabajo necesario durante la ejecución normal, coordina el proceso de punto de sincronización y proporciona información de estado de unidad de trabajo durante el reinicio del subsistema.

El soporte de participantes RRS de IBM MQ for z/OS permite que las aplicaciones de IBM MQ en el lote, TSO y entornos de procedimientos almacenados de Db2 puedan actualizar recursos tanto de IBM MQ como ajenos a IBM MQ (por ejemplo, Db2) dentro de una unidad de trabajo lógica individual. Para obtener información sobre el soporte de participantes de RRS, consulte *z/OS MVS Programming: Resource Recovery*.

La aplicación de IBM MQ puede utilizar MQCMIT y MQBACK o las llamadas RRS equivalentes, SRRCMIT y SRRBACK. Consulte "El adaptador de proceso por lotes RRS" en la página 914 para obtener más información.

Disponibilidad de RRS

Si RRS no está activo en su sistema z/OS, cualquier llamada IBM MQ emitida desde un programa enlazado con cualquier apéndice de RRS (CSQBRSTB o CSQBRRSI) devuelve MQRC_ENVIRONMENT_ERROR.

Procedimientos almacenados de Db2

Si utiliza procedimientos almacenados de Db2 con RRS, tenga en cuenta lo siguiente:

- Los procedimientos almacenados de Db2 que utilizan RRS deben gestionarse mediante el gestor de carga de trabajo (gestionados por WLM).
- Si un procedimiento almacenado gestionado por Db2 contiene llamadas IBM MQ y está enlazado con cualquier apéndice de RRS (CSQBRSTB o CSQBRRSI), la llamada MQCONN o MQCONNX devuelve MQRC_ENVIRONMENT_ERROR.

- Si un procedimiento almacenado gestionado por WLM contiene llamadas IBM MQ y está enlazado con un apéndice que no sea de RRS, la llamada MQCONN o MQCONNX devuelve MQRC_ENVIRONMENT_ERROR, a menos que sea la primera llamada IBM MQ ejecutada desde que se inició el espacio de direcciones del procedimiento almacenado.
- Si el procedimiento almacenado de Db2 contiene llamadas IBM MQ y está enlazado con un apéndice que no sea de RRS, los recursos de IBM MQ actualizados en dicho procedimiento almacenado no se confirman hasta que finalice el espacio de direcciones del procedimiento almacenado, o hasta que un procedimiento almacenado posterior realice un MQCMIT (utilizando un apéndice TSO o por lotes de IBM MQ).
- Es posible que se ejecuten varias copias del mismo procedimiento almacenado de forma simultánea en el mismo espacio de direcciones. Asegúrese de que su programa esté codificado de manera reentrante si desea que Db2 utilice una sola copia del procedimiento almacenado. De lo contrario, es posible que reciba MQRC_HCONN_ERROR en cualquier llamada IBM MQ del programa.
- No codifique MQCMIT ni MQBACK en un procedimiento almacenado de Db2 gestionado por WLM.
- Diseñe todos los programas para que se ejecuten en Language Environment (LE).

IBM i *Puntos de sincronización en CICS para las aplicaciones de IBM i*

IBM MQ for IBM i participa en CICS para las unidades de trabajo de IBM i. Puede utilizar MQI dentro de una aplicación de CICS para IBM i para poner y obtener mensajes dentro de la unidad de trabajo actual.

Puede utilizar el mandato EXEC CICS SYNCPOINT para establecer un punto de sincronización que incluya las operaciones de IBM MQ for IBM i. Para restituir todos los cambios al punto de sincronización anterior, puede utilizar el mandato EXEC CICS SYNCPOINT ROLLBACK.

Si utiliza MQPUT, MQPUT1 o MQGET con la opción MQPMO_SYNCPOINT o MQGMO_SYNCPOINT establecida en una aplicación de CICS para IBM i, no puede cerrar la sesión de CICS para IBM i hasta que IBM MQ for IBM i haya eliminado su registro como un recurso de compromiso de API. Confirme o restituya las operaciones put o get pendientes antes de desconectarse del gestor de colas. Esto permite cerrar la sesión de CICS para IBM i.

Multi *Puntos de sincronización en IBM MQ for Multiplatforms*

El soporte de puntos de sincronización opera en dos tipos de unidades de trabajo: local y global.

Una unidad de trabajo de *local* es aquella en la que los únicos recursos actualizados son los del gestor de colas de IBM MQ. En este caso la coordinación de los puntos de sincronización es proporcionada por el propio gestor de colas utilizando un procedimiento de confirmación en una sola fase.

Una unidad de trabajo *global* es aquella en la que también se actualizan recursos pertenecientes a otros gestores de recursos, tales como bases de datos. El propio IBM MQ puede coordinar tales unidades de trabajo. También pueden ser coordinadas por un controlador de confirmación externo. Por ejemplo:

- Otro gestor de transacciones
- IBM i El controlador de confirmación de IBM i

Para asegurar la integridad total de los datos, utilice un procedimiento de confirmación en dos fases. La confirmación en dos fases puede ser proporcionada por gestores de transacciones y bases de datos compatibles con XA. Por ejemplo:

- TXSeries
- UDB
- IBM i El controlador de confirmación de IBM i

ALW Los productos de IBM MQ pueden coordinar unidades de trabajo globales utilizando un proceso confirmación en dos fases.

IBM i IBM MQ for IBM i puede actuar como un gestor de recursos para unidades de trabajo globales dentro de un entorno WebSphere Application Server, pero no puede actuar como un gestor de transacción.

Punto de sincronismo implícito

Cuando se colocan mensajes persistentes en una cola, IBM MQ está optimizado para colocar mensajes persistentes bajo un punto de sincronización. Si hay varias aplicaciones que colocan mensajes persistentes en la misma cola, su rendimiento es mejor si esas aplicaciones utilizan un punto de sincronización. Esto es debido a que existe menos contienda por la cola si se utiliza un punto de sincronización para colocar mensajes persistentes.

ImplSyncOpenOutput agrega un punto de sincronización implícito cuando las aplicaciones colocan mensajes persistentes fuera del punto de sincronización. Esto proporciona una mejora del rendimiento, sin que las aplicaciones tengan conocimiento del punto de sincronización implícito.

El punto de sincronización implícito sólo proporciona una mejora del rendimiento cuando hay varias aplicaciones que colocan mensajes en una cola, pues reduce la contienda por la cola. Así, **ImplSyncOpenOutput** especifica el número mínimo de aplicaciones que tienen una cola abierta para la salida antes de que se añada un punto de sincronización implícito. El valor predeterminado es 2. Esto significa que, si no especifica **ImplSyncOpenOutput**, el punto de sincronismo implícito sólo se añade si varias aplicaciones están colocando en la cola.

Consulte Parámetros de ajuste para obtener más información.

Unidades locales de trabajo en Multiplatforms
Las unidades de trabajo que solo implican al gestor de colas se llaman unidades de trabajo *locales*. El propio gestor de colas proporciona coordinación de punto de sincronización (coordinación interna) usando un proceso de confirmación en una sola fase.

Para iniciar una unidad de trabajo local, la aplicación emite las peticiones MQGET, MQPUT o MQPUT1 que especifican la opción de punto de sincronización adecuada. La unidad de trabajo se confirma utilizando MQCMIT o se retrotrae utilizando MQBACK. Sin embargo, la unidad de trabajo también finaliza cuando se interrumpe la conexión entre la aplicación y el gestor de colas, de forma intencionada o no intencionada.

Si una aplicación se desconecta (MQDISC) de un gestor de colas mientras una unidad de trabajo global coordinada por IBM MQ sigue activa, se realiza un intento de confirmar la unidad de trabajo. Sin embargo, si la aplicación termina sin desconectarse, la unidad de trabajo se retrotraerá, ya que se considera que la aplicación ha terminado de forma anómala.

Unidades globales de trabajo en Multiplatforms
Una unidad de trabajo global se usa cuando también se necesita incluir actualizaciones a recursos que pertenecen a otros gestores de recursos.

Aquí la coordinación puede ser interna o externa al gestor de colas:

Coordinación de punto de sincronización interna

La coordinación del gestor de colas de unidades de trabajo globales no está soportada por IBM MQ for IBM i o IBM MQ for z/OS. No está soportada en un entorno IBM MQ MQI client.

Aquí, IBM MQ realiza la coordinación. Para iniciar una unidad de trabajo global, la aplicación emite la llamada MQBEGIN.

Como entrada a la llamada MQBEGIN, debe proporcionar el descriptor de conexión (*Hconn*) que devuelve la llamada MQCONN o MQCONNX. Este descriptor representa la conexión con el gestor de colas IBM MQ.

La aplicación emite las peticiones MQGET, MQPUT o MQPUT1 que especifican la opción de punto de sincronización adecuada. Esto significa que se puede utilizar MQBEGIN para iniciar una unidad de trabajo global que actualice recursos locales, recursos que pertenezcan a otros gestores de recursos, o ambos. Las actualizaciones efectuadas a recursos que pertenezcan a otros gestores de recursos se efectúan a través del API de dichos gestores. Sin embargo, no se puede utilizar la MQI para actualizar colas que

pertenezcan a otros gestores de colas. Emita MQCMIT o MQBACK antes de iniciar más unidades de trabajo (locales o globales).

La unidad de trabajo global se confirma con MQCMIT; esto inicia una confirmación en dos fases de todos los gestores de recursos implicados en la unidad de trabajo. Se utiliza un proceso de confirmación en dos fases por el cual, en primer lugar, se solicita a los gestores de recursos (por ejemplo, los gestores de bases de datos compatibles con XA como, por ejemplo, Db2, Oracle y Sybase) que se preparan para confirmar. Solo se les solicitará confirmar cuando estén todos preparados. Si cualquier gestor de recursos indicase que no puede confirmar, se solicitaría una restitución a todos ellos. De forma alternativa, se puede utilizar MQBACK para retrotraer las actualizaciones de todos los gestores de recursos.

Si una aplicación se desconecta (MQDISC) mientras una unidad de trabajo global sigue activa, la unidad de trabajo se confirma. Sin embargo, si la aplicación termina sin desconectarse, la unidad de trabajo se retrotraerá, ya que se considera que la aplicación ha terminado de forma anómala.

La salida de MQBEGIN consiste en un código de terminación y un código de razón.

Cuando se utiliza MQBEGIN para iniciar una unidad de trabajo global, se incluyen todos los gestores de recursos externos configurados en el gestor de colas. Sin embargo, la llamada inicia una unidad de trabajo y termina con una advertencia si:

- No hay gestores de recursos participantes (es decir, no se han configurado ningún gestor de recursos en el gestor de colas).

o

- Uno o más gestores de recursos no están disponibles.

En estos casos, la unidad de trabajo tiene que incluir actualizaciones solo a los gestores de recursos que estaban disponibles al iniciarse la unidad de trabajo.

Si uno de los gestores de recursos no puede confirmar sus actualizaciones, se indicará a todos los gestores de recursos que retrotraigan sus actualizaciones y MQCMIT terminará con un aviso. En circunstancias excepcionales (normalmente, la intervención del operador), una llamada MQCMIT podría fallar si algunos gestores de recursos confirman sus actualizaciones, pero otros la retrotraen; se considera que el trabajo se ha completado con un resultado *mixto*. Tales situaciones se diagnostican en el registro de errores del gestor de colas para que puedan emprenderse acciones correctivas.

Un MQCMIT de una unidad de trabajo global se realiza correctamente si todos los gestores de recursos implicados confirman sus actualizaciones.

Para obtener una descripción de la llamada MQBEGIN, consulte MQBEGIN.

Coordinación de punto de sincronización externa

Esto se produce cuando se ha seleccionado un coordinador de punto de sincronización distinto de IBM MQ; por ejemplo, CICS, Encina o Tuxedo.

En esta situación, los sistemas IBM MQ for AIX, Linux, and Windows registran su interés en el resultado de la unidad de trabajo con el coordinador de punto de sincronismo para que puedan confirmar o retrotraer las operaciones get o put no confirmadas según sea necesario. El coordinador de punto de sincronización externo determina si se proporcionan protocolos de confirmación en una o dos fases.

Cuando se utiliza un coordinador externo, no se pueden emitir MQCMIT, MQBACK ni MQBEGIN. Las llamadas a estas funciones fallan con el código de razón MQRC_ENVIRONMENT_ERROR.

La forma en la que se inicia una unidad de trabajo coordinada externamente depende de la interfaz de programación proporcionada por el coordinador de punto de sincronización. Puede que se necesite una llamada explícita. Si se requiere una llamada explícita y emite una llamada MQPUT especificando la opción MQPMO_SYNCPOINT sin haberse iniciado una unidad de trabajo, se devolverá el código de terminación MQRC_SYNCPOINT_NOT_AVAILABLE.

El ámbito de la unidad de trabajo está determinado por el coordinador de punto de sincronización. El estado de la conexión existente entre la aplicación y el gestor de colas afecta al éxito o al fallo de las llamadas MQI emitidas por una aplicación, no al estado de la unidad de trabajo. Una aplicación puede,

por ejemplo, desconectarse y volver a conectarse con un gestor de colas durante una unidad de trabajo activa, y llevar a cabo más operaciones MQGET y MQPUT dentro de la misma unidad de trabajo. Esto se conoce como desconexión pendiente.

Puede utilizar llamadas de API IBM MQ en programas CICS, independientemente de si opta por utilizar las prestaciones XA de CICS. Si no utiliza XA, las colocaciones y las obtenciones de mensajes en las colas no se gestionarán en unidades de trabajo atómicas de CICS. Un motivo para elegir este método sería que la coherencia global de la unidad de trabajo no fuera importante.

Si la integridad de las unidades de trabajo es importante, hay que usar XA. Cuando utilice XA, CICS utiliza un protocolo de confirmación en dos fases para asegurarse de que todos los recursos de la unidad de trabajo se actualizan juntos.

Para obtener más información sobre cómo configurar el soporte transaccional, consulte Escenarios de soporte transaccional y, también, la documentación de TXSeries CICS, por ejemplo, la publicación *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Multi *Punto de sincronismo implícito en Multiplatforms*

El soporte de punto de sincronación implícito permite colocaciones de mensajes persistentes fuera de un punto de sincronización.

Cuando se colocan mensajes persistentes en una cola, IBM MQ está optimizado para colocar mensajes persistentes bajo un punto de sincronización. Varias aplicaciones que están colocando mensajes persistentes de forma concurrente en la misma cola suelen tener mejor rendimiento cuando utilizan punto de sincronización. Esto se debe a que la estrategia de bloqueo de IBM MQ es más eficiente, si se utiliza el punto de sincronización cuando se colocan mensajes persistentes.

El parámetro **ImplSyncOpenOutput** del archivo `qm.ini` controla si se puede añadir un punto de sincronización implícito cuando las aplicaciones colocan mensajes persistentes fuera de un punto de sincronización. Esto puede proporcionar una mejora en el rendimiento sin que las aplicaciones sean conscientes del punto de sincroníación implícito.

El punto de sincronización implícito solo proporciona un aumento de rendimiento cuando hay varias aplicaciones que están colocando de forma concurrente en una cola, ya que reduce la contención de bloqueos. **ImplSyncOpenOutput** especifica el número mínimo de aplicaciones que tienen una cola abierta para la salida antes de que se pueda añadir un punto de sincronización implícito. El valor predeterminado es *2*. Esto significa que, si no especifica explícitamente **ImplSyncOpenOutput**, el punto de sincronización implícito solo se añade si hay varias aplicaciones que están colocando en la cola.

Si se añade un punto de sincronización implícito, las estadísticas reflejan dicho suceso y puede que se vea una salida de transacción de **runmqsc display conn**.

Configure **ImplSyncOpenOutput**=*OFF* si desea que nunca se añada un punto de sincronización implícito.

Consulte Parámetros de ajuste para obtener más información.

Interfaces para gestores de puntos de sincronismo externos en Multiplatforms

IBM MQ for Multiplatforms da soporte a la coordinación de transacciones mediante gestores de puntos de sincronismo externos que utilizan la interfaz X/Open XA.

Algunos gestores de transacciones XA (TXSeries) requieren que cada gestor de recursos XA proporcione su nombre. Esta es la serie `name` de la estructura de conmutadores XA.

- ALW El gestor de recursos de IBM MQ en AIX, Linux, and Windows se denomina MQSeries_XA_RMI.
- IBM i Para IBM i, el nombre del gestor de recursos es MQSeries XA RMI.

Para obtener más información sobre las interfaces XA consulte el documento XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicado por The Open Group.

En una configuración XA, IBM MQ for Multiplatforms realiza el rol de un gestor de recursos de XA. Un coordinador de puntos de sincronismo XA puede gestionar un conjunto de gestores de recursos XA y sincronizar la confirmación o restitución de transacciones en ambos gestores de recursos. Un gestor de recursos registrado de forma estática funciona de este modo:

1. Una aplicación notifica al coordinador de puntos de sincronismo que desea iniciar una transacción.
2. El coordinador de puntos de sincronismo emite una llamada a cualquier gestor de recursos que conoce para informarles de la transacción actual.
3. La aplicación emite llamadas para actualizar los recursos que gestionan los gestores de recursos asociados a la transacción actual.
4. La aplicación solicita al coordinador de puntos de sincronismo que confirme o retotraiga la transacción.
5. El coordinador de puntos de sincronismo llama a cada gestor de colas utilizando protocolos de confirmación en dos fases para completar la transacción, como se ha solicitado.

La especificación XA requiere que cada gestor de recursos proporcione una estructura denominada conmutador XA. Esta estructura declara las prestaciones del gestor de recursos y las funciones que ha de invocar el coordinador de puntos de sincronismo.

Existen dos versiones de esta estructura:

Tabla 128. Versiones del conmutador XA

Versión	**Descripción**
MQRMIXASwitch	Gestión de recursos XA estática
MQRMIXASwitchDynamic	Gestión de recursos XA dinámica

Para obtener una lista de las bibliotecas que contienen esta estructura, consulte la sección La estructura de conmutación IBM MQ XA.

El coordinador define el método que se debe utilizar para enlazar con un coordinador de puntos de sincronismo XA. Consulte la documentación de dicho coordinador para determinar cómo habilitar IBM MQ para que colabore con su coordinador de puntos de sincronismo XA.

La estructura *xa_info* que se pasa el coordinador de puntos de sincronismo en cualquier llamada *xa_open* puede ser el nombre del gestor de colas que se ha de administrar. Su formato es el mismo que el del nombre del gestor de colas que se ha pasado a MQCONN o MQCONNX, y puede estar en blanco si se va a utilizar el gestor de colas predeterminado. No obstante, puede utilizar los dos parámetros adicionales TPM y AXLIB

TPM le permite especificar el nombre del gestor de transacciones en IBM MQ, por ejemplo, CICS. AXLIB le permite especificar el nombre de la biblioteca real en el gestor de transacciones donde están ubicados los puntos de entrada XA AX.

Si utiliza cualquiera de estos parámetros o un gestor de colas no predeterminado, debe especificar el nombre del gestor de colas utilizando el parámetro QMNAME. Para obtener más información, consulte la sección Los parámetros CHANNEL, TRPTYPE, CONNAME y QMNAME de la serie xa_open.

Restricciones

1. No se permiten unidades de trabajo globales con un Hconn compartido, como se describe en la sección “Conexiones (independientes de hebra) compartidas con MQCONNX” en la página 758.
2. IBM i IBM MQ for IBM i no da soporte al registro dinámico de gestores de recursos XA.

   El único gestor de transacciones soportado es WebSphere Application Server.
3. Windows En los sistemas Windows, todas las funciones declarados en el conmutador XA se declaran como funciones _cdecl.

4. Un coordinador de puntos de sincronismo externo solo puede administrar un gestor de colas cada vez. Esto es debido a que el coordinador tiene una conexión efectiva con cada gestor de colas y, por lo tanto, está sujeto a la regla que solo permite una conexión cada vez.

   Nota: Nota: una aplicación de cliente JMS (aplicación CLIENT JEE) que se ejecuta en un servidor JEE no tiene esta restricción, de modo que una única transacción gestionada por el servidor JEE puede coordinar varios gestores de colas en la misma transacción. No obstante, una aplicación de servidor JMS que se ejecute en modo de enlaces, continúa estando sujeta a la regla que solo permite una conexión cada vez.
5. Todas las aplicaciones que se ejecutan utilizando el coordinador de puntos de sincronismo solo se pueden conectar al gestor de colas que administra el coordinador, debido a que ya están conectados de forma efectiva a dicho gestor de colas. Deben emitir MQCONN o MQCONNX para obtener un descriptor de conexión y deben emitir MQDISC antes de la salida. De forma alternativa, puede utilizar la salida UE014015 para TXSeries CICS.

IBM i *Interfaces para el gestor de puntos de sincronismo externo de IBM i*

IBM MQ for IBM i puede utilizar el control de compromiso de IBM i como un coordinador de puntos de sincronismo externo.

No se permiten las conexiones independientes de la hebra (compartidas) con el control de confirmación. Consulte la publicación *IBM i Programming: Guía de copia de seguridad y recuperación, SC21-8079* para obtener más información acerca de las funciones de control de confirmación de IBM i.

Para iniciar las funciones de control de confirmación de IBM i, utilice el mandato del sistema STRCMTCTL. Para finalizar el control de confirmación, utilice el mandato ENDCMTCTL del sistema.

Nota: El valor predeterminado del *ámbito de definición de confirmación* es *ACTGRP. Esto se debe definir como *JOB en IBM MQ para IBM i. Por ejemplo:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i también puede realizar unidades de trabajo locales que solo contienen actualizaciones para los recursos de IBM MQ. La selección de las unidades de trabajo locales y la participación en las unidades de trabajo globales coordinadas por IBM i se lleva a cabo en cada aplicación, cuando la aplicación invoca MQPUT, MQPUT1 o MQGET, especificando MQPMO_SYNCPOINT, MQGMO_SYNCPOINT o MQBEGIN. Si el control de confirmación no está activo la primera vez que se emite la llamada, IBM MQ inicia una unidad de trabajo local y todas las unidades de trabajo adicionales para esta conexión con IBM MQ también utilizan las unidades de trabajo locales, independientemente de si el control de confirmación se inicia a continuación. Para confirmar una unidad de trabajo local, utilice MQCMIT. Para restituir una unidad de trabajo, utilice MQBACK. Las llamadas de confirmación o restitución de IBM i, tales como el mandato CL COMMIT no tienen ningún efecto en las unidades de trabajo locales de IBM MQ.

Si desea utilizar IBM MQ for IBM i con el control de confirmación nativo de IBM i como un coordinador de puntos de sincronismo externo, asegúrese de que cualquier trabajo con control de confirmación esté activo y que está utilizando IBM MQ en un trabajo de una sola hebra. Si invoca MQPUT, MQPUT1 o MQGET, especificando MQPMO_SYNCPOINT o MQGMO_SYNCPOINT, en un trabajo de varias hebras en el que se ha iniciado el control de confirmación, la llamada falla con el código de razón MQRC_SYNCPOINT_NOT_AVAILABLE.

Se puede utilizar las unidades de trabajo locales y las llamadas MQCMIT y MQBACK en un trabajo de varias hebras.

Si invoca MQPUT, MQPUT1 o MQGET, especificando MQPMO_SYNCPOINT o MQGMO_SYNCPOINT, después de iniciar el control de confirmación, IBM MQ for IBM i se añade como un recurso de confirmación de API a la definición de la confirmación. Normalmente, esto se lleva a cabo en la primera llamada de un trabajo. Mientras haya algún recurso de confirmación de API registrado bajo una definición de confirmación concreta, no podrá finalizar el control de confirmación para dicha definición.

IBM MQ for IBM i elimina su registro como recurso de confirmación de API cuando se desconecta del gestor de colas, si no hay operaciones MQI pendientes en la unidad de trabajo actual.

Si se desconecta del gestor de colas mientras hay operaciones MQPUT, MQPUT1 o MQGET pendientes en la unidad de trabajo, IBM MQ for IBM i continúa registrado como recurso de confirmación de API, para que se le notifique la siguiente confirmación o retrotracción. Cuando se alcanza el siguiente punto de sincronismo, IBM MQ for IBM i confirma o retrotrae los cambios, según sea necesario. Una aplicación se puede desconectar y reconectar con un gestor de colas durante una unidad de trabajo activa y realizar operaciones MQGET y MQPUT adicionales en la misma unidad de trabajo (esto es una desconexión pendiente).

Si intenta emitir un mandato del sistema ENDCMTCTL para esta definición de compromiso, se emite el mensaje CPF8355, lo que indica que hay cambios pendientes activos. Este mensaje también aparece en las anotaciones de trabajo cuando el trabajo finaliza. Para evitarlo, confirme o retrotraiga todas las operaciones de IBM MQ for IBM i pendientes y desconéctese del gestor de colas. De este modo, si se utilizan los mandatos COMMIT o ROLLBACK antes que ENDCMTCTL, el control de fin de confirmación se puede completar correctamente.

Cuando utiliza el control de confirmación de IBM i como un coordinador de puntos de sincronismo externo, no puede emitir las llamadas MQCMIT, MQBACK y MQBEGIN. Las llamadas a estas funciones fallan con el código de razón MQRC_ENVIRONMENT_ERROR.

Para confirmar o retrotraer (esto es, restituir) la unidad de trabajo, utilice uno de los siguientes lenguajes de programación que dan soporte al control de confirmación. Por ejemplo:

- Mandatos CL: COMMIT y ROLLBACK
- Funciones de programación ILE C: _Rcommit y _Rrollback
- ILE RPG: COMMIT y ROLBK
- COBOL/400: COMMIT y ROLLBACK

Cuando utiliza el control de confirmación de IBM i como un coordinador de puntos de sincronismo externo con IBM MQ for IBM i, IBM i realiza un protocolo de confirmación de dos fases en el que participa IBM MQ. Dado que cada unidad de trabajo se confirma en dos fases, es posible que el gestor de colas no esté disponible en la segunda fase dado que ha optado para la confirmación en la primera fase. Esto puede suceder si, por ejemplo, han finalizado los trabajos internos del gestor de colas. En este caso, el registro del trabajo que realiza la confirmación contiene el mensaje CPF835F que indica que ha fallado una operación de confirmación o restitución. Los mensajes anteriores a este indican la causa de problema, si se ha producido durante una operación de confirmación o restitución y también el ID de unidad de trabajo lógica (LUWID) de la unidad de trabajo que ha fallado.

Si el problema es debido a un error del recurso de confirmación de API de IBM MQ durante la confirmación o restitución de una unidad de trabajo preparada, puede utilizar el mandato para completar la operación WRKMQMTRN y restaurar la integridad de la transacción. El mandato requiere que conozca el LUWID de la unidad de trabajo que se ha de confirmar o restituir.

Inicio de aplicaciones de IBM MQ utilizando desencadenantes

Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

Algunas aplicaciones de IBM MQ que dan servicio a las colas se ejecutan continuamente, por lo que siempre están disponibles para recuperar mensajes que llegan a las colas. Sin embargo, es posible que no desee que esto suceda cuando el número de mensajes que llega a las colas es imprevisible. En este caso, las aplicaciones pueden consumir recursos del sistema, aunque no haya mensajes que recuperar.

IBM MQ proporciona un recurso que permite iniciar automáticamente una aplicación cuando hay mensajes disponibles para su recuperación. Este recurso se denomina *desencadenamiento*.

Para obtener información sobre el desencadenamiento de canales, consulte Desencadenamiento de canales.

¿Qué son los desencadenantes?

El gestor de colas define determinadas condiciones como constitutivas de *sucesos desencadenantes*.

Si el descencadenamiento está habilitado para una cola y se produce un suceso desencadenante, el gestor de colas envía un *mensaje desencadenante* a una cola llamada *cola de inicio*. La presencia del mensaje desencadenante en la cola de inicio indica que se ha producido un suceso desencadenante.

Los mensajes desencadenantes generados por el gestor de colas no son persistentes. Esto reduce el registro (lo que mejora el rendimiento) y minimiza los duplicados durante el reinicio, lo que mejora el tiempo de reinicio.

El programa que procesa la cola de inicio se denomina *aplicación supervisora desencadenante*, y su función es leer el mensaje desencadenante y realizar las acciones adecuadas, basándose en la información contenida en el mensaje desencadenante. Normalmente, esta acción consiste en iniciar alguna otra aplicación para de procesar la cola que ha generado el mensaje desencadenante. Desde el punto de vista del gestor de colas, no hay nada especial en la aplicación supervisora desencadenante; es simplemente otra aplicación que lee mensajes de una cola (la cola de inicio).

Si el desencadenamiento está habilitado para una cola, puede crear un *objeto de definición de proceso* asociado con ella. Este objeto contiene información sobre la aplicación que procesa el mensaje que originó el suceso desencadenante. Si se crea el objeto de definición de proceso, el gestor de colas extrae esta información y la coloca en el mensaje desencadenante, para que la utilice la aplicación supervisora desencadenante. El nombre de la definición de proceso asociada con una cola lo proporciona el atributo de la cola local *ProcessName*. Cada cola puede especificar una definición de proceso diferente o pueden compartir la misma definición de proceso entre varias colas.

Si desea desencadenar el inicio de un canal, no es necesario que defina un objeto de definición de proceso. En su lugar, utilice la definición de cola de transmisión.

Los clientes de IBM MQ que se ejecutan en AIX, Linux, and Windows dan soporte al desencadenamiento. Una aplicación que se ejecuta en un entorno de cliente es la misma que la que se ejecuta en un entorno de IBM MQ completo, excepto que se enlaza con las bibliotecas de cliente. No obstante, el supervisor desencadenante y la aplicación que va a iniciarse deben estar en el mismo entorno.

El desencadenamiento consta de los elementos siguientes:

Cola de aplicación
Una *cola de aplicación* es una cola local que, cuando se ha establecido el desencadenamiento y se cumplen las condiciones, requiere que se graben los mensajes desencadenantes.

Definición de proceso
Una cola de aplicación puede tener un *objeto de definición de proceso* asociado, que contiene los detalles de la aplicación que va a obtener mensajes de la cola de aplicación. (Consulte Atributos para definiciones de proceso para obtener una lista de atributos).

Recuerde que si desea que un desencadenante inicie un canal, no es necesario definir una definición de proceso objeto.

Cola de transmisión
Necesita una cola de transmisión si desea un desencadenante para iniciar un canal.

Para una cola de transmisión en una plataforma distinta de Linux, el atributo *TriggerData* de la cola de transmisión puede especificar el nombre del canal que se va a iniciar. Esto puede sustituir la definición de proceso para desencadenar canales, pero solo se utiliza cuando no se crea una definición de proceso.

Suceso desencadenante
Un *suceso desencadenante* es un suceso que hace que el gestor de colas genere un mensaje desencadenante. Normalmente, se trata de un mensaje que llega a una cola de aplicación, pero también puede generarse en otras ocasiones. Por ejemplo, consulte “Condiciones para un suceso desencadenante” en la página 893.

IBM MQ tiene un rango de opciones que permiten controlar las condiciones que provocan un suceso desencadenante (consulte “Control de sucesos desencadenantes” en la página 898).

Mensaje de desencadenante
El gestor de colas crea un *mensaje de desencadenante* cuando reconoce un suceso desencadenante. Copia en el mensaje de desencadenante información sobre la aplicación que debe iniciarse. Esta

información procede de la cola de aplicación y el objeto de definición de proceso asociado con la cola de aplicación.

Los mensajes de desencadenante tienen un formato fijo (consulte “Formato de los mensajes desencadenantes” en la página 905).

Cola de inicio

Una *cola de inicio* es una cola local en la que el gestor de colas coloca mensajes de desencadenante. Tenga en cuenta que una cola de inicio no puede ser una cola alias o una cola de modelo.

Un gestor de colas puede poseer más de una cola de inicio, y cada una está asociada con una o varias colas de aplicación.

z/OS Una cola compartida, una cola local accesible para los gestores de colas de un grupo de compartición de colas, puede ser una cola de inicio en IBM MQ for z/OS.

Supervisor desencadenante

Un *supervisor desencadenante* es un programa que se ejecuta continuamente y da servicio a una o varias colas de inicio. Cuando llega un mensaje de desencadenante a una cola de inicio, el supervisor desencadenante recupera el mensaje. El supervisor desencadenante utiliza la información del mensaje de desencadenante. Emite un mandato para iniciar la aplicación que va a recuperar los mensajes que llegan a la cola de aplicación y le pasa la información contenida en la cabecera del mensaje de desencadenante, que incluye el nombre de la cola de aplicación.

En todas las plataformas, un supervisor desencadenante especial conocido como el iniciador de canal es el responsable de iniciar los canales.

z/OS En z/OS, el iniciador de canal suele iniciarse manualmente, aunque puede iniciarse automáticamente cuando se inicia un gestor de colas cambiando CSQINP2 en el JCL de arranque del gestor de colas.

Multi En Multiplatforms, el iniciador de canal se inicia automáticamente cuando se inicia el gestor de colas, o puede iniciarse manualmente con el mandato **runmqchi**.

Para obtener más información, consulte “Proceso de cola de inicio por parte de supervisores desencadenantes” en la página 901.

Para entender cómo funciona el mecanismo de desencadenamiento, consulte la Figura 95 en la página 889, que es un ejemplo de tipo desencadenante FIRST (MQTT_FIRST).

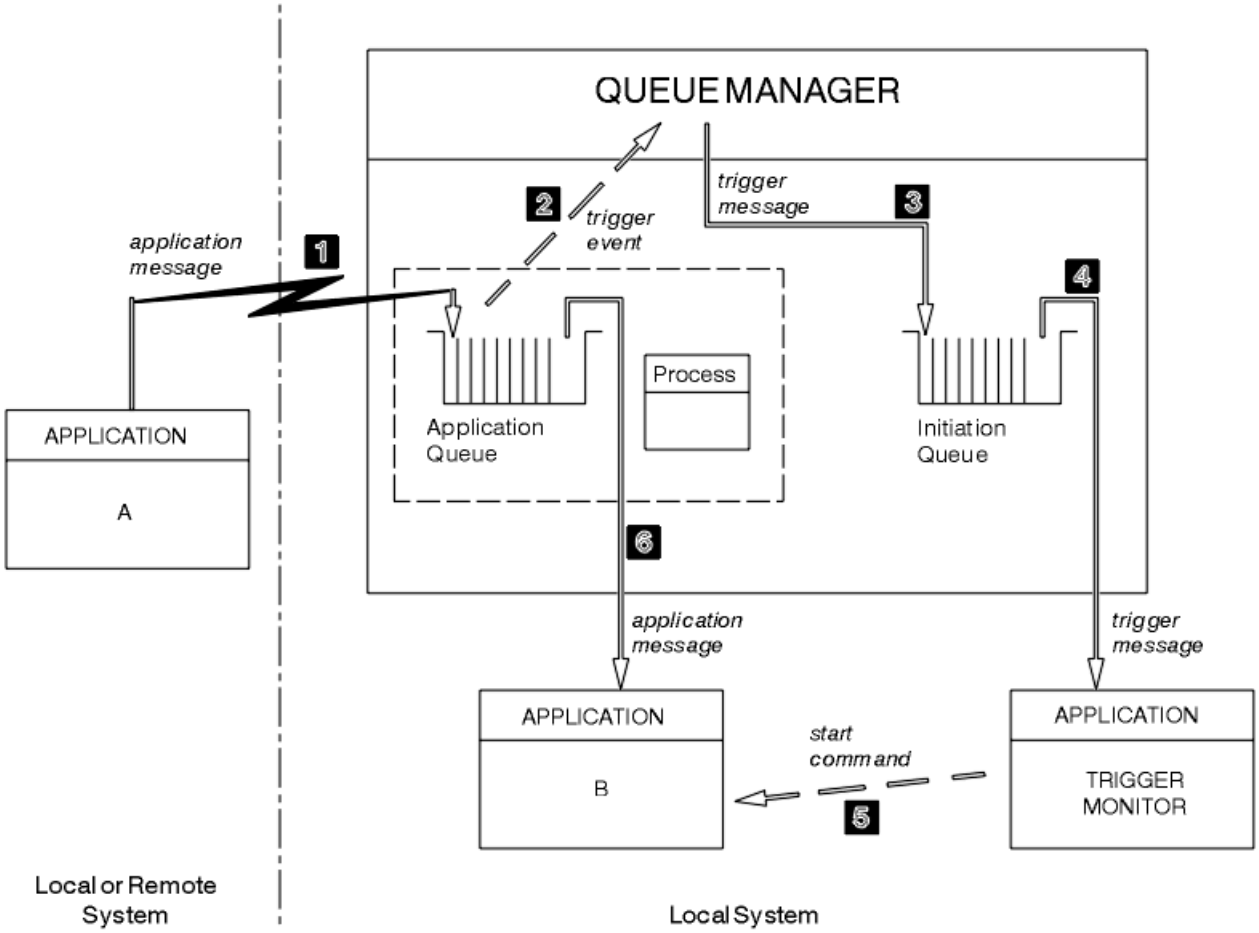


Figura 95. Flujo de aplicaciones y mensajes desencadenantes

En la Figura 95 en la página 889, la secuencia de sucesos es:

1. La aplicación A, que puede ser local o remota para el gestor de colas, coloca un mensaje en la cola de aplicación. No hay ninguna aplicación que tenga esta cola abierta para realizar entradas. Sin embargo, este hecho solo es relevante para el tipo de desencadenante FIRST y DEPTH.
2. El gestor de colas comprueba para ver si se cumplen las condiciones con las que se tiene que generar un suceso desencadenante. Se cumplen y se genera un suceso desencadenante. La información contenida en el objeto de definición de proceso asociado se utiliza cuando se crea el mensaje desencadenante.
3. El gestor de colas crea un mensaje desencadenante y lo coloca en la cola de inicio asociada con esta cola de aplicación, pero solo si una aplicación (supervisor desencadenante) tiene la cola de inicio abierta para realizar entradas.
4. El supervisor desencadenante recupera el mensaje desencadenante de la cola de inicio.
5. El supervisor desencadenante emite un mandato para iniciar la aplicación B (la aplicación de servidor).
6. La aplicación B abre la cola de aplicación y recupera el mensaje.

Nota:

1. Si la cola de aplicación está abierta para realiza una entrada con cualquier programa y tiene establecido el desencadenante para FIRST o DEPTH, no se produce ningún suceso desencadenante porque ya se está atendiendo la cola.
2. Si la cola de inicio no está abierta para realizar entradas, el gestor de colas no genera ningún mensaje desencadenante; espera hasta que una aplicación abra la cola de inicio para la entrada.

3. Cuando utilice el desencadenamiento de canales, utilice el tipo de desencadenante FIRST o DEPTH.
4. Las aplicaciones desencadenadas se ejecutan con el ID de usuario y el grupo del usuario que inició el supervisor desencadenante, el usuario de CICS o el usuario que inició el gestor de colas.

Hasta ahora, la relación entre las colas del desencadenamiento ha sido una relación solo de uno a uno. Consulte la Figura 96 en la página 890.

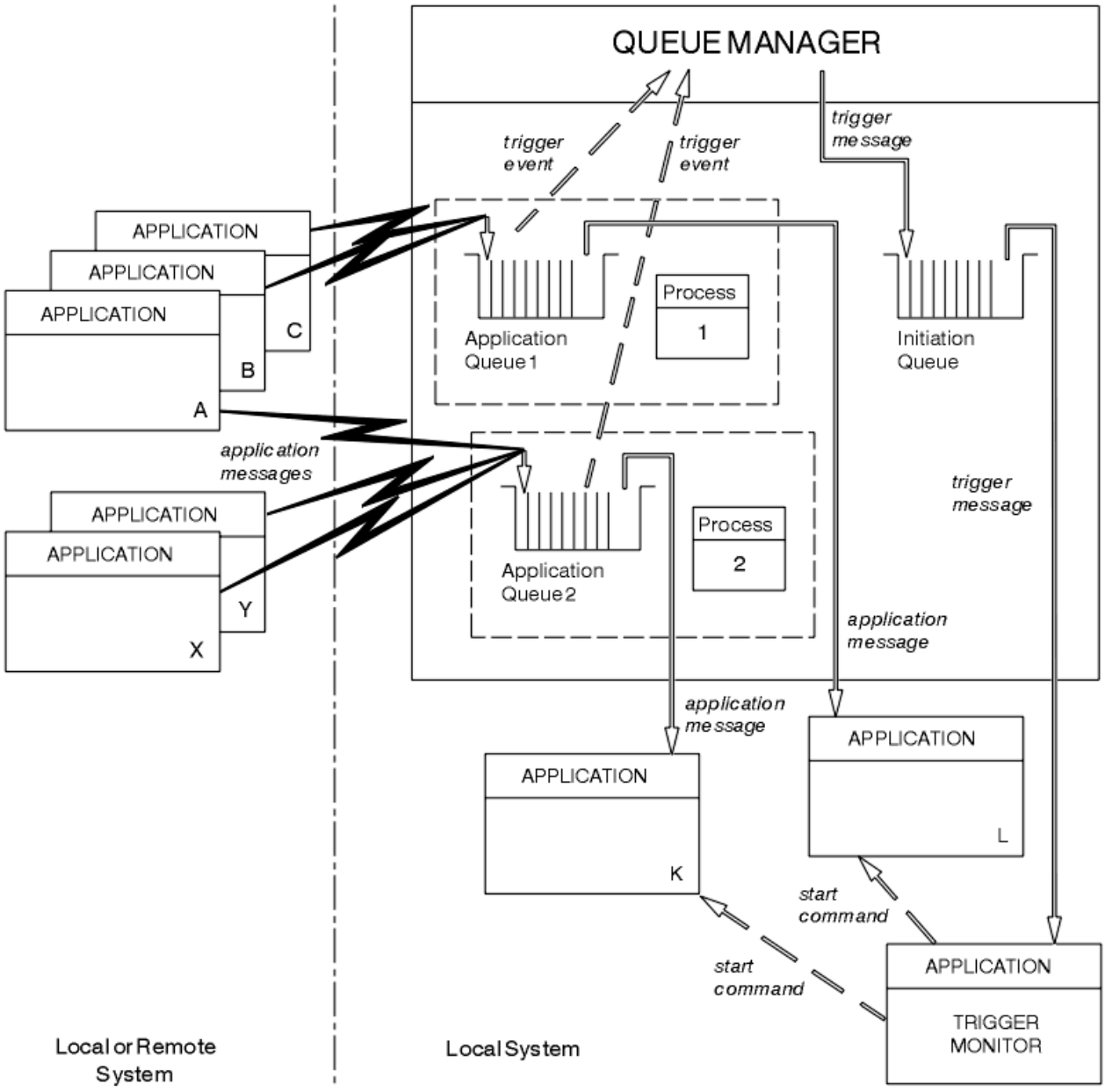


Figura 96. Relación de colas dentro del desencadenante

Una cola de aplicación tiene un objeto de definición de proceso asociado que contiene los detalles de la aplicación que procesará el mensaje. El gestor de colas coloca la información en el mensaje desencadenante, de modo que solo es necesaria una cola de inicio. El supervisor desencadenante extrae esta información del mensaje desencadenante e inicia la aplicación correspondiente para encargarse del mensaje en cada cola de aplicación.

Recuerde que, si desea desencadenar el inicio de un canal, no es necesario que defina un objeto de definición de proceso. La definición de cola de transmisión puede determinar el canal que se va a desencadenar.

Utilice los enlaces siguientes para obtener más información sobre el inicio de aplicaciones de IBM MQ utilizando desencadenantes:

- "Requisitos previos del desencadenamiento" en la página 891
- "Condiciones para un suceso desencadenante" en la página 893
- "Control de sucesos desencadenantes" en la página 898
- "Diseño de una aplicación que utiliza las colas desencadenadas" en la página 900
- "Proceso de cola de inicio por parte de supervisores desencadenantes" en la página 901
- "Propiedades de los mensajes desencadenantes" en la página 905
- "Cuando el desencadenamiento no funciona" en la página 906

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Apertura y cierre de objetos" en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

"Colocación de mensajes en una cola" en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Cómo trabajar con clústeres y MQI" en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

Requisitos previos del desencadenamiento

Utilice esta información para conocer los pasos que se deben realizar antes de utilizar el desencadenamiento.

Para que la aplicación pueda aprovechar el desencadenamiento, siga estos pasos:

1. O bien:
   a. Cree una cola de inicio para la cola de aplicación. Por ejemplo:

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   o

b. Determine el nombre de una cola local que exista y que pueda utilizarse en la aplicación (normalmente, este nombre es SYSTEM.DEFAULT.INITIATION.QUEUE o, si está iniciando canales con desencadenantes, SYSTEM.CHANNEL.INITQ) y especifique su nombre en el campo *InitiationQName* de la cola de aplicación.

2. Asocie la cola de inicio con la cola de aplicación. Un gestor de colas puede tener más de una cola de inicio. Puede que desee que algunas de sus colas de aplicación estén servidas por distintos programas, en cuyo caso, puede utilizar una cola de inicio para cada programa de servicio, aunque no es necesario. A continuación, se muestra un ejemplo de cómo crear una cola de aplicación:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i A continuación, se proporciona un extracto de un programa CL para IBM MQ for IBM i que crea una cola de inicio:

```
/*   Queue used by AMQSINQA                                        */
           CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                       QTYPE(*LCL)  REPLACE(*YES)                  +
                       MQMNAME                                     +
                       TEXT('queue for AMQSINQA')                  +
                       SHARE(*YES)                /* Shareable   */+
                       DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                   +
                       TRGENBL(*YES)  /* Trigger control on      */+
                       TRGTYPE(*FIRST)/* Trigger on first message*/+
                       PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. Si está desencadenando una aplicación, cree un objeto de definición de proceso para que contenga información relativa a la aplicación que dará servicio a la cola de aplicación. Por ejemplo, para desencadenar-iniciar una transacción de nómina de CICS llamada PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i A continuación, se proporciona un extracto de un programa CL para IBM MQ for IBM i que crea un objeto de definición de proceso:

```
/*   Process definition                                            */
           CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       REPLACE(*YES)                               +
                       MQMNAME                                     +
                       TEXT('trigger process for AMQSINQA')        +
                       ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                       APPID('AMQSINQA')    /* Program name      */
```

Cuando el gestor de colas crea un mensaje de desencadenante, copia información de los atributos del objeto de definición de proceso en el mensaje de desencadenante.

Plataforma	Para crear un objeto de definición de proceso
AIX, Linux, and Windows sistemas	Utilice DEFINE PROCESS o utilice SYSTEM.DEFAULT.PROCESS y modifíquelo utilizando ALTER PROCESS
z/OS z/OS	Utilice DEFINE PROCESS (consulte el código de ejemplo en el paso "3" en la página 892), o utilice las operaciones y los paneles de control.
IBM i IBM i	Utilice un programa CL que contenga código como en el paso "3" en la página 892.

4. Opcional: cree una definición de cola de transmisión y utilice espacios en blanco para el atributo **ProcessName**.

   El atributo **TrigData** puede contener el nombre del canal que se va a desencadenar o puede dejarse en blanco. Excepto en IBM MQ for z/OS, si se deja en blanco, el iniciador de canal busca los archivos de definición de canal hasta que encuentra un canal asociado con la cola de transmisión especificada. Cuando el gestor de colas crea un mensaje desencadenante, copia la información del atributo **TrigData** de la definición de cola de transmisión en el mensaje de desencadenante.

5. Si ha creado un objeto de definición de proceso para especificar las propiedades de la aplicación que va a dar servicio a la cola de aplicación, asocie el objeto de proceso con la cola de aplicación nombrándola en el atributo **ProcessName** de la cola.

Plataforma	Utilizar los mandatos
AIX, Linux, and Windows sistemas	ALTER QLOCAL
z/OS z/OS	ALTER QLOCAL
IBM i IBM i	CHGMQMQ

6. Las instancias de inicio los supervisores desencadenantes IBM i (o los servidores desencadenantes en IBM MQ for IBM i) que van a dar servicio a las colas de inicio que ha definido. Para obtener más información, consulte "Proceso de cola de inicio por parte de supervisores desencadenantes" en la página 901.

Si desea conocer los mensajes de desencadenante sin entregar, asegúrese de que el gestor de colas tenga definida una cola de mensajes no entregados. Especifique el nombre de la cola en el campo del gestor de colas *DeadLetterQName*.

A continuación, puede establecer las condiciones de desencadenante que necesita, utilizando los atributos del objeto de cola que define la cola de aplicación. Para obtener más información, consulte "Control de sucesos desencadenantes" en la página 898.

Condiciones para un suceso desencadenante

Un gestor de colas crea un mensaje desencadenante cuando se cumplen las condiciones detalladas en este tema.

Las referencias a colas compartidas en este tema significan las colas compartidas de un grupo de compartición de colas, solo disponibles en IBM MQ for z/OS.

Las condiciones siguientes hacen que el gestor de colas cree un mensaje desencadenante:

1. Se *coloca* un mensaje en una cola.

2. El mensaje tiene una prioridad mayor que o igual al umbral de la prioridad de desencadenamiento de la cola. Esta prioridad se configura en el atributo de cola local **TriggerMsgPriority**; si se establece a cero, todos los mensajes cumplirán la condición.
3. El número de mensajes de la cola que tengan una prioridad mayor o igual que *TriggerMsgPriority* era anteriormente, dependiendo de *TriggerType*:
   - Cero (para el tipo de desencadenante MQTT_FIRST)
   - Cualquier número (para el tipo de desencadenante MQTT_EVERY)
   - *TriggerDepth* menos 1 (para el tipo de desencadenante MQTT_DEPTH)

   Nota:

   a. En las colas locales no compartidas, el gestor de colas cuenta los mensajes confirmados y los no confirmados cuando evalúa si existen las condiciones para que se genere un suceso desencadenante. Por tanto, una aplicación podría iniciarse cuando no tenga mensajes que recuperar, porque los mensajes de la cola aún no se hayan confirmado. En esta situación, considere la posibilidad de utilizar la opción wait con un valor *WaitInterval* adecuado, de modo que la aplicación espere a que lleguen sus mensajes.

   b. En las colas locales compartidas, el gestor de colas solo cuenta los mensajes confirmados.
4. En los tipos de desencadenante FIRST o DEPTH, ningún programa tiene la cola de aplicación abierta para eliminar los mensajes (es decir, el atributo de cola local **OpenInputCount** es cero).

   Nota:

   a. En las colas compartidas, se aplican condiciones especiales cuando varios gestores de colas tienen supervisores desencadenantes en ejecución contra una cola. En esta situación, si uno o varios gestores de colas tienen la cola abierta para entrada compartida, los criterios desencadenantes en los otros gestores de colas se tratan como *TriggerType* MQTT_FIRST y *TriggerMsgPriority* cero. Cuando todos los gestores de colas cierran la cola para no permitir la entrada, las condiciones desencadenantes vuelven a ser las condiciones especificadas en la definición de cola.

      Un escenario de ejemplo afectado por esta condición son varios gestores de colas QM1, QM2y QM3 con un supervisor desencadenante en ejecución para una cola de aplicación A. Un mensaje llega a A que cumple las condiciones para desencadenarse y se genera un mensaje desencadenante en la cola de inicio. El supervisor desencadenante en QM1 obtiene el mensaje desencadenante y desencadena una aplicación. La aplicación desencadenada abre la cola de aplicación para permitir la entrada compartida. A partir de este punto, las condiciones desencadenantes para la cola de aplicación A se evalúan como *TriggerType* MQTT_FIRST, y *TriggerMsgPriority* cero en los gestores de colas QM2 y QM3, hasta que QM1 cierre la cola de aplicación.

   b. En las colas compartidas, esta condición se aplica por cada gestor de colas. Es decir, el *OpenInputCount* de un gestor de colas tiene que ser cero para que se el gestor de colas genere un mensaje desencadenante para la cola. Sin embargo, si algún gestor de colas del grupo de compartición de colas tiene la cola abierta utilizando la opción MQOO_INPUT_EXCLUSIVE, no se genera ningún mensaje de desencadenante para dicha cola por parte de ninguno de los gestores de colas del grupo de compartición de colas.

      El cambio en la forma en que se evalúan las condiciones desencadenantes tiene lugar cuando la aplicación desencadenada abre la cola para permitir la entrada. En aquellos escenarios en los que solo hay un supervisor desencadenante en ejecución, otras aplicaciones pueden tener el mismo efecto, porque también abren la cola de aplicación para la entrada. No importa si la cola de aplicación es abierta por una aplicación iniciada por un supervisor desencadenante o por alguna otra aplicación; lo que provoca un cambio en los criterios de desencadenamiento es el hecho de que la cola esté abierta para permitir la entrada en otro gestor de colas.
5. En IBM MQ for z/OS, si la cola de aplicación es la cola con un atributo **Usage** de MQUS_NORMAL, las solicitudes de obtención para la misma no están inhibidas (es decir, el atributo de cola **InhibitGet** es MQQA_GET_ALLOWED). Además, si la cola de aplicación desencadenada tiene un atributo **Usage** MQUS_XMITQ, las peticiones de obtención que reciba no se inhibirán.

6. O bien:
   - El atributo de cola local **ProcessName** de la cola no está en blanco y se ha creado el objeto de definición de proceso identificado por dicho atributo, o
   - El atributo de cola local **ProcessName** de la cola está en blanco, pero la cola es de transmisión. Puesto que la definición de proceso es opcional, el atributo **TriggerData** también puede contener el nombre del canal por iniciar. En este caso, el mensaje desencadenante contiene atributos con los valores siguientes:
     - **QName**: nombre de cola
     - **ProcessName**: blancos
     - **TriggerData**: datos desencadenantes
     - **ApplType**: MQAT_UNKNOWN
     - **ApplId**: blancos
     - **EnvData**: blancos
     - **UserData**: blancos
7. Se ha creado una cola de inicio y se ha especificado en el atributo de cola local **InitiationQName**. Además:
   - Las peticiones de obtención no están inhibidas en la cola de inicio (es decir, el valor del atributo de cola **InhibitGet** es MQQA_GET_ALLOWED).
   - Las peticiones de obtención dirigidas a la cola de inicio no deben inhibirse (es decir, el valor del atributo de cola **InhibitPut** tiene que ser MQQA_PUT_ALLOWED).
   - El valor del atributo **Usage** de la cola de inicio tiene que ser MQUS_NORMAL.
   - En entornos en los que soporten las colas dinámicas, la cola de inicio no puede ser una cola dinámica que se haya marcado como borrada lógicamente.
8. Actualmente, un supervisor desencadenante tiene la cola de inicio abierta para poder eliminar mensajes (es decir, el atributo de cola local **OpenInputCount** es mayor que cero).
9. El control de desencadenante (atributo de cola local **TriggerControl**) de la cola de aplicación se establece a MQTC_ON. Para ello, establezca el atributo **trigger** cuando defina la cola o utilice el comando ALTER QLOCAL.
10. El tipo de desencadenante (atributo de cola local **TriggerType**) no es MQTT_NONE.

    Si se cumplen todas las condiciones necesarias y el mensaje que ha provocado que la condición de desencadenante se coloque como parte de una unidad de trabajo, el mensaje de desencadenamiento no estará disponible para su recuperación por parte de la aplicación del supervisor de desencadenante mientras no se complete la unidad de trabajo, tanto si la unidad de trabajo se confirma como, en el caso del tipo de desencadenante MQTT_FIRST o MQTT_DEPTH, se restituye.
11. Un mensaje adecuado se coloca en la cola, con un atributo **TriggerType** MQTT_FIRST o MQTT_DEPTH, y la cola:
    - No estaba previamente vacía (MQTT_FIRST), o bien
    - Tenía **TriggerDepth** o más mensajes (MQTT_DEPTH)

    y se cumplen las condiciones “2” en la página 894 a “10” en la página 895 (excluyendo “3” en la página 894), si, en el caso de MQTT_FIRST, ha transcurrido un intervalo suficiente (atributo de gestor de colas **TriggerInterval**) desde la escritura del último mensaje desencadenante de esta cola.

    Con esto se tiene en cuenta el servidor de colas que termina antes de procesar todos los mensajes de la cola. La finalidad del intervalo de desencadenante es reducir el número de mensajes de desencadenante duplicados que se generan.

    Nota: Si se para y reinicia el gestor de colas, se restablece el temporizador **TriggerInterval**. Hay una pequeña ventana durante la cual es posible generar dos mensajes de desencadenante. La ventana existe cuando el atributo de desencadenante de la cola se establece a ENABLED al

mismo tiempo que llega un mensaje y la cola no estaba vacía previamente (MQTT_FIRST) o tenía **TriggerDepth** o más mensajes (MQTT_DEPTH).

12. La única aplicación que da servicio a una cola emite una llamada MQCLOSE, para un **TriggerType** MQTT_FIRST o MQTT_DEPTH, y hay al menos:

    - Un (MQTT_FIRST), o
    - **TriggerDepth** (MQTT_DEPTH)

    también se cumplen los mensajes en la cola de prioridad suficiente (condición "2" en la página 894) y las condiciones "6" en la página 895 a "10" en la página 895 .

    Con esto se tiene en cuenta un servidor de colas que emite una llamada MQGET, encuentra la cola vacía y, por tanto, termina; no obstante, en el intervalo entre las llamadas MQGET y MQCLOSE, llegan uno o más mensajes.

    Nota:

    a. Si el programa que da servicio a la cola de aplicación no recupera todos los mensajes, esto puede provocar un bucle cerrado. Cada vez que el programa cierra la cola, el gestor de colas crea otro mensaje desencadenante que hace que el supervisor de desencadenante vuelva a iniciar el programa de servidor.
    b. Si el programa que da servicio a la cola de aplicación restituye su solicitud de obtención (o si el programa termina de forma anómala) antes de cerrar la cola, ocurre lo mismo. No obstante, si el programa cierra la cola antes de restituir la solicitud de obtención y, por otro lado, la cola está vacía, no se crea ningún mensaje desencadenante.
    c. Para evitar que se genere este tipo de bucle, utilice el campo *BackoutCount* de MQMD para detectar los mensajes que se restituyan repetidamente. Para obtener más información, consulte "Mensajes que se restituyen" en la página 47.

13. Se cumplen las condiciones siguientes usando MQSET o un comando:

    a. • **TriggerControl** se cambia a MQTC_ON, o
       • **TriggerControl** ya es MQTC_ON y se cambia el valor de **TriggerType**, **TriggerMsgPriority** o **TriggerDepth** (si procede),

       y hay al menos:

       - Un (MQTT_FIRST o MQTT_EVERY), o
       - **TriggerDepth** (MQTT_DEPTH)

       mensajes en la cola de prioridad suficiente (condición "2" en la página 894) y condiciones "4" en la página 894 a "10" en la página 895 (excluyendo "8" en la página 895) también están satisfechos.

       Con esto se tiene en cuenta una aplicación o un operador que cambia el criterio de desencadenante cuando ya se han cumplido las condiciones para que se genere un desencadenante.

    b. El valor del atributo de cola **InhibitPut** de una cola de inicio cambia de MQQA_PUT_INHIBITED a MQQA_PUT_ALLOWED y hay al menos:

       - Un (MQTT_FIRST o MQTT_EVERY), o
       - **TriggerDepth** (MQTT_DEPTH)

       mensajes de suficiente prioridad (condición "2" en la página 894) en cualquiera de las colas de las que esta es la cola de inicio y también se cumplen las condiciones "4" en la página 894 a "10" en la página 895. (Se genera un mensaje desencadenante por cada una de tales colas que cumpla las condiciones).

       Con esto se tienen en cuenta los mensajes de desencadenante que no se generan debido a la condición MQQA_PUT_INHIBITED en la cola de inicio, aunque ahora esta condición se haya cambiado.

c. El valor del atributo de cola **InhibitGet** de una cola de aplicación cambia de MQQA_GET_INHIBITED a MQQA_GET_ALLOWED y hay al menos:

- Un (MQTT_FIRST o MQTT_EVERY), o
- **TriggerDepth** (MQTT_DEPTH)

mensajes de suficiente prioridad (condición “2” en la página 894) en la cola y también se cumplen las condiciones “4” en la página 894 a “10” en la página 895, excluyendo “5” en la página 894.

Esto permite que las aplicaciones solo se desencadenen cuando puedan recuperar mensajes de la cola de aplicación.

d. Una aplicación de supervisor desencadenante emite una llamada MQOPEN para entrada desde una cola de inicio, y hay al menos:

- Un (MQTT_FIRST o MQTT_EVERY), o
- **TriggerDepth** (MQTT_DEPTH)

mensajes de suficiente prioridad (condición “2” en la página 894) en cualquiera de las colas de aplicación de las que esta es cola de inicio, y también se cumplen las condiciones “4” en la página 894 a “10” en la página 895 (excluyendo “8” en la página 895), y ninguna otra aplicación tiene la cola de inicio abierta para la entrada (se genera un mensaje desencadenante por cada una de las colas que cumplen las condiciones).

Con esto se tienen en cuenta los mensajes que llegan a las colas mientras no está ejecutando el supervisor desencadenante y el gestor de colas se reinicia y se pierden los mensajes de desencadenante (que no son persistentes).

14. MSGDLVSQ se ha establecido correctamente. Si configura MSGDLVSQ=FIFO, los mensajes se entregan en la cola conforme al orden "primero en entrar, primero en salir" (FIFO). La prioridad del mensaje se ignora y se asigna al mensaje la prioridad predeterminada de la cola. Si se establece **TriggerMsgPriority** a un valor superior a la prioridad predeterminada de la cola, no se desencadena ningún mensaje. Si se establece **TriggerMsgPriority** a un valor igual o menor que la prioridad predeterminada de la cola, se produce el desencadenamiento para los tipos FIRST, EVERY y DEPTH. Para obtener información sobre estos tipos, consulte la descripción del campo **TriggerType** en “Control de sucesos desencadenantes” en la página 898.

Si establece MSGDLVSQ=PRIORITY y la prioridad del mensaje es igual o mayor que el campo *TriggerMsgPriority* , los mensajes sólo cuentan para un suceso desencadenante. En este caso, se produce el desencadenamiento en los tipos FIRST, EVERY y DEPTH. Por ejemplo, si se transfieren 100 mensajes de prioridad inferior a la de **TriggerMsgPriority**, la profundidad de cola efectiva, a efectos de desencadenamiento, sigue siendo cero. Si luego se coloca otro mensaje en la cola, pero esta vez la prioridad es mayor o igual que **TriggerMsgPriority**, la profundidad de cola aumenta de cero a uno y se cumple la condición de **TriggerType** FIRST.

Notas:

1. En el paso “12” en la página 896 (donde los mensajes de desencadenante se generan como resultado de algún suceso distinto de la llegada de un mensaje a la cola de aplicación), el mensaje desencadenante no se coloca como parte de una unidad de trabajo. Además, si **TriggerType** es MQTT_EVERY y si hay uno o más mensajes en la cola de aplicación, solo se genera un mensaje desencadenante.
2. Si IBM MQ segmenta un mensaje durante un MQPUT, no se procesará un evento de desencadenante mientras no se hayan colocado correctamente todos los segmentos en la cola. Sin embargo, una vez que los segmentos de mensajes están en la cola, IBM MQ los tratará como mensajes individuales a efectos de desencadenante. Por ejemplo, un solo mensaje lógico dividido en tres partes hace que solo se procese un suceso desencadenante la primera vez que se le hace el MQPUT y se segmenta. Sin embargo, cada uno de los tres segmentos provoca el procesamiento de sus propios sucesos desencadenantes a medida que se desplazan por la red IBM MQ network.
3. Para IBM MQ for z/OS, si se configura una cola compartida para el desencadenamiento y se pierde la conexión con el recurso de acoplamiento que aloja la cola compartida, es posible que se genere un suceso desencadenante y que se transfiera un mensaje a la cola de inicio. Esto puede suceder

incluso cuando no se ha colocado ningún mensaje en la configuración de cola compartida original para el desencadenamiento. Esto se debe a la sobreindicación de bits por parte de la macro IXLVECTR tal como se documenta en El vector de notificación de lista.

Control de sucesos desencadenantes

Los sucesos desencadenantes se controlan utilizando algunos de los atributos que definen la cola de aplicación. Esta información también proporciona ejemplos de utilización de los tipos de desencadenante: EVERY, FIRST y DEPTH.

Puede habilitar e inhabilitar el desencadenamiento, y puede seleccionar el número o la prioridad de los mensajes que cuentan para un suceso desencadenante. En Atributos de objetos se proporciona una descripción completa de estos atributos.

Los atributos pertinentes son:

TriggerControl
Utilice este atributo para habilitar e inhabilitar el desencadenamiento de una cola de aplicación.

TriggerMsgPriority
La prioridad mínima que un mensaje debe tener para contar para un suceso desencadenante. Si un mensaje de prioridad menor que *TriggerMsgPriority* llega a la cola de aplicación, el gestor de colas ignora el mensaje cuando determina si debe crear un mensaje desencadenante. Si *TriggerMsgPriority* se establece en cero, todos los mensajes cuentan para un suceso desencadenante.

TriggerType
Además del tipo de desencadenante NONE (que inhabilita el desencadenamiento al igual que el valor *TriggerControl* en OFF), puede utilizar los siguientes tipos de desencadenante para establecer la sensibilidad de una cola para desencadenar sucesos:

EVERY
Se produce un suceso desencadenante cada vez que un mensaje llega a la cola de la aplicación. Utilice este tipo de desencadenante si desea que se inicien varias instancias de una aplicación.

FIRST
Se produce un suceso desencadenante únicamente cuando el número de mensajes en la cola de la aplicación cambia de cero a uno. Utilice este tipo de desencadenante si desea que un programa de servicio se inicie cuando el primer mensaje llega a una cola, continúe hasta que no hay más mensajes para el proceso y luego finalice. Siempre debe procesar la cola hasta que esté vacía. Consulte también “Caso especial de tipo de desencadenante FIRST” en la página 899.

DEPTH
Se produce un suceso desencadenante únicamente cuando el número de mensajes en la cola de aplicación alcanza el valor del atributo **TriggerDepth**. Un uso típico de este tipo de desencadenamiento es iniciar un programa cuando se reciben todas las respuestas a un conjunto de solicitudes.

Desencadenamiento por profundidad: Con el desencadenamiento por profundidad, el gestor de colas inhabilita el desencadenamiento (utilizando el atributo *TriggerControl*) después de crear un mensaje desencadenante. La aplicación debe volver a habilitar el desencadenamiento ella misma (utilizando la llamada MQSET) después de que esto haya sucedido.

La acción de inhabilitar el desencadenamiento no está bajo el control del punto de sincronismo, de modo que el mecanismo de desencadenamiento no se puede volver a habilitar mediante la restitución de una unidad de trabajo. Si un programa restituye una solicitud de transferencia (put) que ha causado un suceso desencadenante, o si el programa termina de forma anómala, debe volver a habilitar el desencadenamiento utilizando la llamada MQSET o el mandato ALTER QLOCAL.

TriggerDepth
El número de mensajes en una cola que provoca un suceso desencadenante cuando se utiliza el desencadenamiento por la profundidad.

Las condiciones que deben cumplirse para que un gestor de colas cree un mensaje desencadenante se describen en “Condiciones para un suceso desencadenante” en la página 893.

Ejemplo de uso del tipo de desencadenante EVERY

Suponga que tiene una aplicación que genera solicitudes de seguros de automóviles. La aplicación puede enviar mensajes de solicitud a un número de compañías de seguros, especificando la misma cola de respuestas cada vez. Podría establecer un desencadenante de tipo EVERY en esta cola de respuestas de forma que cada vez que llega una respuesta, la respuesta podría desencadenar una instancia del servidor para procesar la respuesta.

Ejemplo de uso del tipo de desencadenante FIRST

Suponga que tiene una organización con una serie de sucursales que transmiten detalles de las transacciones comerciales diarias a la oficina central. Todas ellas lo hacen al mismo tiempo, al final de la jornada laboral, y en la oficina hay una aplicación que procesa los detalles de todas las sucursales. El primer mensaje en llegar a la oficina podría provocar un suceso desencadenante que inicia esta aplicación. Esta aplicación podría continuar el proceso hasta que no haya más mensajes en la cola.

Ejemplo de uso del tipo de desencadenante DEPTH

Suponga que tiene una aplicación de una agencia de viajes que crea una sola solicitud para confirmar una reserva de vuelo, para confirmar una reserva para una habitación de hotel, para alquilar un coche y solicitar algunos cheques de viaje. La aplicación puede separar estos elementos en cuatro mensajes de solicitud, enviando cada uno a un destino independiente. Podría establecer un desencadenante de tipo de DEPTH en la cola de respuestas (con la profundidad establecida en el valor 4), de forma que se reinicie únicamente cuando las cuatro respuestas hayan llegado.

Si otro mensaje (posiblemente de una solicitud distinta) llega a la cola de respuestas antes de la última de las cuatro respuestas, la aplicación solicitante se desencadena pronto. Para evitarlo, cuando utilice el desencadenamiento DEPTH para recopilar varias respuestas a una solicitud, debe utilizar siempre una nueva cola de respuestas para cada solicitud.

Caso especial de tipo de desencadenante FIRST

Con el tipo de desencadenante FIRST, si ya existe un mensaje en la cola de la aplicación cuando llega otro mensaje, el gestor de colas no suele crear otro mensaje desencadenante.

Sin embargo, puede que la aplicación que atiende la cola no abra en realidad la cola (por ejemplo, la aplicación podría finalizar, posiblemente debido a un problema del sistema). Si se ha colocado un nombre de aplicación incorrecto en el objeto de definición de proceso, la aplicación de servicio de la cola no recopilará ninguno de los mensajes. En estas situaciones, si otro mensaje llega a la cola de aplicación, no hay ningún servidor en ejecución para procesar este mensaje (y cualquier otro mensaje en la cola).

Para solucionar este problema, el gestor de colas crea más mensajes desencadenantes en los casos siguientes:

- Si otro mensaje llega a la cola de aplicación, pero sólo si ha transcurrido un intervalo de tiempo predefinido desde que el gestor de colas creó el último mensaje desencadenante para esa cola. Este intervalo de tiempo se define en el atributo del gestor de colas *TriggerInterval*. El valor predeterminado es 999 999 999 milisegundos.
- En IBM MQ for z/OS, las colas de aplicación que nombran una cola de inicio de apertura se exploran de forma periódica. Si se ha pasado *TRIGINT* milisegundos desde que se envió el último mensaje desencadenante y la cola satisface las condiciones para un suceso desencadenante y CURDEPTH es mayor que cero, se genera un mensaje desencadenante. Este proceso se denomina desencadenamiento de seguridad.

Tenga en cuenta los puntos siguientes cuando tenga que decidir un valor para el intervalo de desencadenante para utilizar en la aplicación:

- Si establece *TriggerInterval* en un valor bajo, y no hay ninguna aplicación que atienda la cola de aplicación, el tipo de desencadenante FIRST puede comportarse como el tipo de desencadenante EVERY. Esto depende de la velocidad a la que se colocan los mensajes en la cola de aplicación, que a su vez depende de otras actividades del sistema. Esto se debe a que, si el intervalo de desencadenante es muy pequeño, se genera otro mensaje desencadenante cada vez que se pone un mensaje en la cola de la aplicación, aunque el tipo desencadenante sea FIRST y no EVERY. (El tipo de desencadenante FIRST con un intervalo de desencadenante de cero es equivalente al tipo de desencadenante EVERY.)
- En IBM MQ for z/OS si establece *TRIGINT* en un valor bajo y no hay ninguna aplicación que sirva la cola de aplicación FIRST de tipo de desencadenante, el desencadenamiento de restitución generará un mensaje desencadenante cada vez que se lleve a cabo la exploración periódica de las colas de aplicaciones que denominan colas de iniciación abiertas.
- Si se restituye una unidad de trabajo (consulte Mensajes de desencadenante y unidades de trabajo) y el intervalo de desencadenante se ha establecido en un valor alto (o en el valor predeterminado), se genera un mensaje desencadenante cuando se restituya la unidad de trabajo. Sin embargo, si ha establecido el intervalo de desencadenante en un valor bajo o en cero (lo que hace que el tipo de desencadenante FIRST se comporte como el tipo de desencadenante EVERY), se pueden generar muchos mensajes desencadenantes. Si se restituye la unidad de trabajo, todos los mensajes desencadenantes seguirán estando disponibles. El número de mensajes desencadenantes que se generan depende del intervalo de desencadenante. Si el intervalo de desencadenante se establece en cero, se genera el número máximo de mensajes.

Diseño de una aplicación que utiliza las colas desencadenadas

Se ha visto cómo configurar y controlar el desencadenamiento de las aplicaciones. He aquí algunos consejos por tener en cuenta al diseñar la aplicación.

Mensajes desencadenantes y unidades de trabajo

Los mensajes desencadenantes creados debido a sucesos desencadenantes que no formen parte de una unidad de trabajo se colocan en la cola de inicio, fuera de cualquier unidad de trabajo, sin dependencia de ningún otro mensaje, y están disponibles para que los recupere el supervisor desencadenante de forma inmediata.

Los mensajes desencadenantes creados debido a sucesos desencadenantes que formen parte de una unidad de trabajo estarán disponibles en la cola de inicio en el momento en que se resuelva la unidad de trabajo, tanto si esta se confirma como si se restituye.

Si el gestor de colas no puede colocar un mensaje desencadenante en una cola de inicio, se colocará en la cola de mensajes no entregados.

Nota:

1. El gestor de colas cuenta los mensajes confirmados y los no confirmados cuando evalúa si se dan las condiciones para que generar un suceso desencadenante.

   Con un desencadenamiento de tipo FIRST o DEPTH, los mensajes desencadenantes están disponibles incluso si se restituye la unidad de trabajo para que un mensaje desencadenante siempre esté disponible cuando se cumplan las condiciones necesarias. Por ejemplo, considere una solicitud de colocación dentro de una unidad de trabajo para una cola que se desencadene con el tipo de desencadenante FIRST. Esto hace que el gestor de colas cree un mensaje desencadenante. Si se produce otra solicitud de colocación desde otra unidad de trabajo, esto no provoca otro suceso desencadenante porque, ahora, el número de mensajes de la cola de aplicación ha cambiado de uno a dos, situación que no cumple las condiciones de un suceso desencadenante. Ahora si se restituye la primera unidad de trabajo, pero se confirma la segunda, se sigue creando un mensaje desencadenante.

   No obstante, esto significa que, a veces, los mensajes desencadenantes se crean cuando no se cumplen las condiciones de un suceso desencadenante. Las aplicaciones que utilizan el desencadenamiento siempre tienen que estar preparadas para poder manejar esta situación. Se recomienda utilizar la opción de espera con la llamada MQGET, estableciendo *WaitInterval* a un valor que resulte adecuado.

Los mensajes desencadenantes creados siempre están disponibles, tanto si se restituye como si se confirma la unidad de trabajo.

2. Para las colas compartidas locales (es decir, las colas compartidas de un grupo de compartición de colas), el gestor de colas solo recuenta mensajes confirmados.

Obtención de mensajes de una cola desencadenada

Al diseñar aplicaciones que utilicen desencadenamientos, tenga en cuenta que podría haber un retardo entre un supervisor desencadenante que inicia un programa y otros mensajes que pasan a estar disponibles en la cola de aplicación. Esto puede ocurrir cuando el mensaje que provoca el suceso desencadenante se confirma antes que los demás.

Para dejar tiempo a que los mensajes lleguen, utilice siempre la opción wait en la llamada MQGET para eliminar los mensajes de una cola para la que se hayan establecido condiciones desencadenantes. El valor de *WaitInterval* (intervalo de espera) tiene que ser suficiente para permitir que transcurra el máximo de tiempo razonable entre la colocación de un mensaje y la confirmación de esa llamada de colocación. Si el mensaje llega de un gestor de colas remoto, este tiempo se ve afectado por:

- El número de mensajes que se transfieren antes de ser confirmados.
- La velocidad y disponibilidad del enlace de comunicaciones.
- Los tamaños de los mensajes.

Para obtener un ejemplo de una situación en la que se tiene que emplear la llamada MQGET con la opción wait, considere el mismo ejemplo usado al describir las unidades de trabajo. El ejemplo trataba de una solicitud de colocación dentro de una unidad de trabajo para una cola que se desencadene con el tipo de desencadenante FIRST. Este suceso hace que el gestor de colas cree un mensaje desencadenante. Si se produce otra solicitud de colocación desde otra unidad de trabajo, no se provoca otro suceso desencadenante, porque el número de mensajes de la cola de aplicación no ha cambiado de cero a uno. Ahora si se restituye la primera unidad de trabajo, pero se confirma la segunda, se sigue creando un mensaje desencadenante. Por tanto, el mensaje desencadenante se crea en el momento en que se restituye la primera unidad de trabajo. Si existe un retardo significativo antes de que se confirme el segundo mensaje, es posible que la aplicación desencadenada tenga que esperar al mismo.

Con un tipo de desencadenante DEPTH, podría producirse un retardo incluso si se llegaran a confirmar todos los mensajes relevantes. Supongamos que el atributo de cola **TriggerDepth** tiene el valor 2. Cuando llegan dos mensajes a la cola, el segundo hace que se cree un mensaje desencadenante. No obstante, si el segundo mensaje fuera el primero en confirmarse, sería en ese momento cuando el mensaje desencadenante pasara a estar disponible. El supervisor desencadenante inicia el programa servidor, pero el programa solo podrá recuperar el segundo mensaje mientras no se confirme el primero. Por tanto, puede que el programa tenga que esperar a que el primer mensaje pase a estar disponible.

Diseñe la aplicación para que termine si no hay ningún mensaje disponible para su recuperación cuando venza el intervalo de espera. Si, más adelante, llegan uno o más mensajes, deje que la aplicación se vuelva a desencadenar para procesarlos. Este método evita que las aplicaciones se queden desocupadas, utilizando recursos innecesariamente.

Proceso de cola de inicio por parte de supervisores desencadenantes

Para un gestor de colas, un supervisor desencadenante es como cualquier otra aplicación que sirve a una cola. Sin embargo, un supervisor desencadenante sirve colas de inicio.

Un supervisor desencadenante es normalmente un programa de ejecución continua. Cuando llega un mensaje desencadenante a una cola de iniciación, el supervisor desencadenante recupera dicho mensaje. Utiliza información en el mensaje para emitir un mandato para iniciar la aplicación que va a procesar los mensajes de la cola de aplicaciones.

El supervisor desencadenante debe pasar información suficiente al programa que está iniciando de forma que el programa pueda realizar las acciones correctas en la cola de aplicaciones correcta.

Un iniciador de canal es un ejemplo de un tipo especial de supervisor desencadenante para agentes de canal de mensajes. Sin embargo, en esta situación, debe utilizar el tipo de desencadenante FIRST o DEPTH.

ALW *Supervisores desencadenantes en sistemas AIX, Linux, and Windows*

Este tema contiene información sobre los supervisores desencadenantes proporcionados en sistemas AIX, Linux, and Windows .

Los supervisores desencadenantes siguientes se proporcionan para el entorno de servidor:

amqstrg0

Este es un supervisor desencadenante de ejemplo que proporciona un subconjunto de la función que proporciona **runmqtrm**. Consulte “Utilización de los programas de ejemplo en Multiplataformas” en la página 1086 para obtener más información sobre amqstrg0.

runmqtrm

La sintaxis de este mandato es **runmqtrm** [*-m QMgrName*] [*-q InitQ*], donde QMgrName es el gestor de colas y InitQ es la cola de inicio. La cola predeterminada es SYSTEM.DEFAULT.INITIATION.QUEUE en el gestor de colas predeterminado. Llama a programas para los mensajes de desencadenante adecuados. Este supervisor desencadenante da soporte al tipo de aplicación predeterminado.

La serie de mandato que pasa el supervisor desencadenante al sistema operativo se crea según se indica a continuación:

1. *ApplId* (ID de aplicación) de la definición PROCESS correspondiente (si se ha creado)
2. La estructura MQTMC2, delimitada por comillas dobles
3. *EnvData* (ID de aplicación) de la definición PROCESS correspondiente (si se ha creado)

donde *ApplId* es el nombre del programa a ejecutar como si se especificara en la línea de mandatos.

El parámetro que se pasa es la estructura de caracteres MQTMC2. Se invoca una serie de mandato que tiene esta serie, exactamente tal como se proporciona, con comillas dobles, para que el mandato del sistema la acepte como un parámetro.

El supervisor desencadenante no comprueba si hay otro mensaje en la cola de inicio hasta que finaliza la aplicación que acaba de iniciarse. Si la aplicación debe mucho que procesar, es posible que el supervisor desencadenante no pueda seguir el ritmo debido al número de mensajes de desencadenante que llegan. Tiene dos opciones:

- Tener más supervisores desencadenantes en ejecución
- Ejecutar las aplicaciones iniciadas en segundo plano

Si tiene más supervisores desencadenantes en ejecución, puede controlar el número máximo de aplicaciones que pueden ejecutarse en cualquier momento. Si ejecuta aplicaciones en segundo plano, no hay ninguna restricción impuesta por IBM MQ en relación al número de aplicaciones que se pueden ejecutar.

Linux AIX Para ejecutar la aplicación iniciada en segundo plano en AIX and Linux, coloque un & al final del *EnvData* de la definición PROCESS.

Para ejecutar la aplicación iniciada en segundo plano en sistemas Windows, dentro del campo *ApplId*, ponga como prefijo el nombre de su aplicación con un mandato START. Por ejemplo:

```
START ?B AMQSECHA
```

Nota: Windows Donde una vía de acceso de Windows tiene espacios como parte del nombre de vía de acceso, estos se deben delimitar con comillas dobles (") para garantizar que se tratan como un único argumento. Por ejemplo, "`C:\Program Files\Application Directory\Application.exe`".

A continuación se muestra un ejemplo de una serie APPLICID donde el nombre de archivo incluye espacios como parte de la vía de acceso:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

La sintaxis del mandato START de Windows en el ejemplo incluye una serie vacía delimitada con comillas dobles. START especifica que el primer argumento entre comillas dobles se tratará como el título del nuevo mandato. Para asegurar que Windows no malinterpreta la vía de acceso de aplicación para un argumento de 'título', añada una serie de título delimitada por comillas dobles al mandato antes del nombre de aplicación.

Los supervisores desencadenantes siguientes se proporcionan para el cliente IBM MQ:

runmqtmc
Se trata del mismo que runmqtrm excepto que enlaza con las bibliotecas de IBM MQ MQI client.

ALW *Supervisor desencadenante para CICS*

El supervisor desencadenante amqltmc0 se proporciona para CICS. Funciona de la misma manera que el supervisor desencadenante estándar, pero se ejecuta de una manera distinta y desencadena transacciones CICS.

Este tema sólo se aplica a sistemas Windows, AIX and Linux x86-64 .

El supervisor desencadenante se proporciona como un programa de CICS; defralo con un nombre de transacción de 4 caracteres. Especifique el nombre de 4 caracteres para iniciar el supervisor desencadenante. Utiliza el gestor de colas predeterminado (tal como se indica en el archivo qm.ini o, en IBM MQ for Windows, el registro) y SYSTEM.CICS.INITIATION.QUEUE.

Si desea utilizar un gestor de colas o una cola diferente, cree la estructura MQTMC2 del supervisor desencadenante: esto requiere que escriba un programa utilizando la llamada EXEC CICS START, debido a que la estructura es demasiado larga para añadirla como parámetro. A continuación, pase la estructura MQTMC2 como dato a la solicitud START del supervisor desencadenante.

Cuando utilice la estructura MQTMC2, necesitará proporcionar solo los parámetros *StrucId*, *Version*, *QName* y **QMgrName** para el supervisor desencadenante, ya que no hace referencia a ningún otro campo.

Los mensajes se leen de la cola de inicio y se utilizan para iniciar transacciones CICS , utilizando EXEC CICS START, suponiendo que APPL_TYPE en el mensaje desencadenante sea MQAT_CICS. La lectura de mensajes de la cola de iniciación se realiza bajo el control de punto de sincronización de CICS.

Se generan mensajes cuando se inicia y se detiene el supervisor, así como cuando se produce un error. Estos mensajes se envían a la cola de datos transitoria CSMT.

Tabla 129. Versiones disponibles del supervisor desencadenante.

Una tabla con dos columnas. La primera columa enumera las versiones disponibles del supervisor desencadenante y la segunda columna muestra las plataformas para las que se utiliza cada versión.

Versión	**Utilización**
amqltmc0	TXSeries para: • AIX AIX • Linux Sistemas Linux x86-64
amqltmc4	Windows TXSeries para Windows 5.1
amqltmcc	Versión vinculada de cliente del supervisor desencadenante de CICS

Si necesita un supervisor desencadenante para otros entornos, escriba un programa que pueda procesar los mensajes de desencadenante que el gestor de colas pone en las colas de iniciación. Dicho programa debe realizar las acciones siguientes:

1. Utilizar la llamada MQGET para esperar a que llegue un mensaje a la cola de iniciación.
2. Examinar los campos de la estructura MQTM del mensaje de desencadenante para buscar el nombre de la aplicación a iniciar y el entorno en el que se ejecuta.
3. Emitir un mandato de inicio específico del entorno.

   z/OS Por ejemplo, en el proceso por lotes de z/OS, envíe un trabajo al lector interno.
4. Convertir la estructura MQTM a la estructura MQTMC2 si es necesario.
5. Pasar la estructura MQTMC2 o MQTM a la aplicación iniciada. Esto puede contener datos de usuario.
6. Asociar con la cola de aplicación la aplicación que debe servir a dicha cola. Puede hacer esto denominando el objeto de definición de proceso (si se ha creado) en el atributo **ProcessName** de la cola. Para nombrar el objeto de definición de proceso, puede utilizar el mandato **DEFINE QLOCAL** o **ALTER QLOCAL** .

   IBM i En IBM i, también puede utilizar CRTMQMQ o CHGMQMQ.

Para obtener más información sobre la interfaz de supervisor desencadenante, consulte MQTMC2.

IBM i *Supervisores de desencadenantes en IBM i*

En IBM i, en lugar del mandato de control de **runmqtrm** , utilice el IBM MQ for IBM i mandato CL **STRMQMTRM**.

Utilice el mandato STRMQMTRM de este modo:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Los detalles son para runmqtrm.

También se proporcionan los siguientes programas de ejemplo que puede utilizar como modelos para escribir sus propios supervisores de desencadenantes:

AMQSTRG4
Este es un supervisor de desencadenante que envía un trabajo de IBM i para el proceso que se ha de iniciar, pero esto significa que existe un proceso adicional asociada a cada mensaje de desencadenante.

AMQSERV4
Este es un servidor de desencadenantes. Para cada mensaje desencadenante, este servidor ejecuta el mandato para el proceso en su propio trabajo y puede invocar transacciones CICS.

Tanto el supervisor de desencadenante como el servidor de desencadenantes pasan una estructura MQTMC2 a los programas que inician. Para obtener una descripción de esta estructura, consulte la sección MQTMC2. Estos dos ejemplos se entregan en formato fuente y formato ejecutable.

Dado que estos supervisores de desencadenantes solo pueden invocar programas IBM i nativos, no pueden desencadenar directamente programas Java, ya que las clases Java se encuentran en IFS. No obstante, los programas Java se pueden desencadenar indirectamente desencadenando un programa CL que, a continuación, invoca el programa Java y pasa por la estructura TMC2. El tamaño mínimo de la estructura TMC2 es de 732 bytes.

El siguiente es código fuente de un CLP de ejemplo:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

El siguiente programa de supervisor de desencadenante se proporciona para el IBM MQ MQI client: RUNMQTMC

Invoque RUNMQTMC de este modo:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

Propiedades de los mensajes desencadenantes

Los siguientes temas describen otras propiedades de los mensajes desencadenantes.

- "Persistencia y prioridad de los mensajes desencadenantes" en la página 905
- "Reinicio del gestor de colas y mensajes desencadenantes" en la página 905
- "Mensajes desencadenantes y atributos de objetos" en la página 905
- "Formato de los mensajes desencadenantes" en la página 905

Persistencia y prioridad de los mensajes desencadenantes

Los mensajes desencadenantes no son persistentes debido a que no es un requisito de los mismos.

No obstante, las condiciones para generar sucesos desencadenantes persisten, por lo tanto se generan mensajes desencadenantes cuando se cumplen estas condiciones. Si se pierde un mensaje desencadenante, dado que el mensaje de aplicación continúa existiendo en la cola de aplicación, se garantiza que el gesto de colas genera un mensaje desencadenante en cuanto se cumplen todas las condiciones.

Si se restituye una unidad de trabajo, siempre se entrega cualquier mensaje desencadenante que se genere.

Los mensajes desencadenantes tienen prioridad predeterminada de la cola de iniciación.

Reinicio del gestor de colas y mensajes desencadenantes

Después del reinicio un gestor de colas, cuando se abre a continuación una cola de iniciación para entrada, se puede colocar el mensaje desencadenante en esta cola de aplicación, si una cola de aplicación asociada incluye mensajes y se ha definido como desencadenante.

Mensajes desencadenantes y atributos de objetos

Los mensajes desencadenantes se crean en función de los valores de los atributos de desencadenante que estén en vigor en el momento en que se produce el suceso desencadenante.

Si el mensaje desencadenante no está disponible para un supervisor de desencadenantes hasta más tarde, debido a que el mensaje que lo ha generado no se ha colocado en una unidad de trabajo, los cambios que se hayan realizado mientras tanto en los atributos de desencadenante no tienen ningún efecto en el mensaje desencadenante. En concreto, inhabilitar el desencadenante no impide que un mensaje desencadenante esté disponible una vez creado. Asimismo, es posible que la cola de aplicación ya no exista en el momento en que el mensaje desencadenante está disponible.

Formato de los mensajes desencadenantes

El formato de un mensaje desencadenante se define mediante la estructura MQTM.

Tiene los campos siguientes, que rellena el gestor de colas cuando crea el mensaje desencadenante, utilizando la información de las definiciones de objetos de la cola de aplicación y de los procesos asociados a dicha cola:

StrucId
El identificador de la estructura.

Version
La versión de la estructura.

QName
El nombre de la aplicación en la que se ha producido el suceso desencadenante. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **QName** de la cola de aplicación.

ProcessName
El nombre del objeto de definición de proceso asociado a la cola de aplicación. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **ProcessName** de la cola de aplicación.

TriggerData
Un campo de formato libre que utiliza el supervisor de desencadenantes. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **TriggerData** de la cola de aplicación. En cualquier producto IBM MQ, salvo IBM MQ for z/OS, este campo se puede utilizar para especificar el nombre del canal que se ha de desencadenar.

ApplType
El tipo de aplicación que ha de iniciar el supervisor de desencadenantes. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **ApplType** del objeto de definición de proceso identificado en *ProcessName*.

ApplId
Una serie de caracteres que identifica la aplicación que ha de iniciar el supervisor de desencadenantes. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **ApplId** del objeto de definición de proceso identificado en *ProcessName*.

Cuando se utiliza el supervisor desencadenante CKTI, proporcionado por CICS, el atributo **ApplId** del objeto de definición de proceso es un identificador de transacción CICS .

Cuando se utiliza CSQQTRMN proporcionado por IBM MQ for z/OS, el atributo **ApplId** del objeto de definición de proceso es un identificador de transacción IMS .

EnvData
Un campo de caracteres que contiene datos relacionados con el entorno para que los utilice supervisor de desencadenantes. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **EnvData** del objeto de definición de proceso identificado en *ProcessName*. El supervisor desencadenante proporcionado por CICS(CKTI) o el supervisor desencadenante proporcionado por IBM MQ for z/OS(CSQQTRMN) no utilizan este campo, pero otros supervisores desencadenantes podrían optar por utilizarlo.

UserData
Un campo de caracteres que contiene datos de usuario para que los utilice supervisor de desencadenantes. Cuando el gestor de colas crea un mensaje desencadenante, rellena este campo utilizando el atributo **UserData** del objeto de definición de proceso identificado en *ProcessName*. Este campo se puede utilizar para especificar el nombre del canal que se ha de desencadenar.

Puede encontrar una descripción completa de la estructura de los mensajes desencadenantes en MQTM.

Cuando el desencadenamiento no funciona

Un programa no se desencadena si el supervisor desencadenante no puede iniciar el programa o el gestor de colas no puede entregar el mensaje desencadenante. Por ejemplo, el identificador de aplicación del objeto de proceso tiene que especificar que el programa se tiene que iniciar en segundo plano; de lo contrario, el supervisor desencadenante no podrá iniciar el programa.

Si se crea un mensaje desencadenante, pero no se puede colocar en la cola de inicio (por ejemplo, porque la cola está llena o la longitud del mensaje es mayor que la longitud máxima de mensaje especificada para la cola de inicio), el mensaje desencadenante se coloca en la cola de mensajes no entregados (mensaje sin entregar).

Si la operación de colocación en la cola de mensajes no entregados no se puede completar satisfactoriamente, se descarta el mensaje de desencadenante y se envía un mensaje de aviso

a la consola z/OS o al operador del sistema o se coloca en el registro de errores.

La colocación del mensaje desencadenante en la cola de mensajes no entregados puede generar un mensaje desencadenante para dicha cola. Este segundo mensaje desencadenante se descarta si añade un mensaje a la cola de mensajes no entregados.

Si el programa se desencadena correctamente, pero termina de forma anómala antes de recibir el mensaje de la cola, utilice una utilidad de rastreo (por ejemplo, CICS AUXTRACE si el programa se está ejecutando bajo CICS) para encontrar la causa del error.

Cómo trabajar con clústeres y MQI

Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

Utilice los enlaces siguientes para obtener más información sobre las opciones disponibles en las llamadas y los códigos de retorno que se usan en clústeres:

- "La llamada MQOPEN y los clústeres" en la página 907
- "MQPUT, MQPUT1 y clústeres" en la página 909
- "MQINQ y clústeres" en la página 909
- "MQSET y clústeres" en la página 910
- "Códigos de retorno" en la página 910

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

"Apertura y cierre de objetos" en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

"Colocación de mensajes en una cola" en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

"Confirmación y restitución de unidades de trabajo" en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

"Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

"Utilización y escritura de aplicaciones en IBM MQ for z/OS" en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

"IMS y las aplicaciones puente IMS en IBM MQ for z/OS" en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

La llamada MQOPEN y los clústeres

La cola en la que se coloca, o de la que se lee, un mensaje cuando se abre una cola de clúster depende de la llamada MQOPEN.

Selección de la cola de destino

Si no se proporciona un nombre de gestor de colas en el descriptor de objeto, MQOD, el gestor de colas seleccionará el gestor de colas al que enviar el mensaje. Si se proporciona un nombre de gestor de colas en el descriptor de objeto, los mensajes siempre se envían al gestor de colas seleccionado.

Si el gestor de colas está seleccionando el gestor de colas de destino, la selección dependerá de las opciones de enlace, MQOO_BIND_*, y de si existe una cola local. Si hay una instancia local de la cola, siempre se abrirá con preferencia sobre una instancia remota, a menos que el atributo CLWLUSEQ esté establecido a ANY. De lo contrario, la selección dependerá de las opciones de enlace.Se debe especificar MQOO_BIND_ON_OPEN o MQOO_BIND_ON_GROUP cuando se utilizan grupos de mensajes con clústeres para asegurarse de que todos los mensajes del grupo se procesan en el mismo destino.

Si el gestor de colas está seleccionando el gestor de colas de destino, lo hace de forma rotativa, utilizando el algoritmo de gestión de carga de trabajo; consulte Equilibrado de la carga de trabajo en un clúster.

Cuando se utiliza el algoritmo de equilibrado de carga de trabajo, aquel dependerá de la forma en que se abra la cola de clúster:

- MQOO_BIND_ON_OPEN: El algoritmo se utiliza una vez en el momento en que la aplicación abre la cola.
- MQOO_BIND_NOT_FIXED -el algoritmo se utiliza para cada mensaje colocado en la cola.
- MQOO_BIND_ON_GROUP -el algoritmo se utiliza una vez al principio de cada grupo de mensajes.

MQOO_BIND_ON_OPEN
La opción MQOO_BIND_ON_OPEN de la llamada MQOPEN especifica que el gestor de colas de destino va a ser fijo. Utilice la opción MQOO_BIND_ON_OPEN si hay varias instancias de la misma cola dentro de un clúster. Todos los mensajes colocados en la cola que especifican el descriptor de objeto devuelto por la llamada MQOPEN se dirigen al mismo gestor de colas.

- Utilice la opción MQOO_BIND_ON_OPEN si los mensajes tienen afinidades. Por ejemplo, si un lote de mensajes tiene que ser procesado íntegramente por el mismo gestor de colas, especifique MQOO_BIND_ON_OPEN cuando abra la cola. IBM MQ fija el gestor de colas y la ruta que se va a seguir por parte de todos los mensajes colocados en dicha cola.
- Si se especifica la opción MQOO_BIND_ON_OPEN, hay que reabrir la cola para que se seleccione una nueva instancia de la misma.

MQOO_BIND_NOT_FIXED
La opción MQOO_BIND_NOT_FIXED de la llamada MQOPEN especifica que el gestor de colas de destino no es fijo. Los mensajes escritos en la cola que especifiquen el descriptor de objeto devuelto por la llamada MQOPEN se direccionan a un gestor de colas en el momento del MQPUT por cada mensaje. Utilice la opción MQOO_BIND_NOT_FIXED si no desea forzar que todos los mensajes se escriban en el mismo destino.

- No especifique MQOO_BIND_NOT_FIXED y MQMF_SEGMENTATION_ALLOWED a la vez. Si lo hace, los segmentos del mensaje se pueden entregar a distintos gestores de colas dispersos por todo el clúster.

MQOO_BIND_ON_GROUP
Permite que una aplicación solicite que un grupo de mensajes se asigne a la misma instancia de destino. Esta opción solo es válida para colas y solo afecta a colas de clúster. Si se especifica en una cola que no sea de clúster, la opción se pasará por alto.

- Los grupos solo se direccionan a un único destino cuando se especifica MQPMO_LOGICAL_ORDER en la MQPUT. Cuando se especifica MQOO_BIND_ON_GROUP, pero un mensaje no forma parte de un grupo lógico, en su lugar se utiliza el comportamiento BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF
Si no especifica MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED o MQOO_BIND_ON_GROUP, la opción predeterminada es MQOO_BIND_AS_Q_DEF. La utilización de MQOO_BIND_AS_Q_DEF hace que el enlace que se utiliza para el manejador de cola se tome del atributo de cola DefBind.

Relevancia de las opciones de MQOPEN

Las opciones de MQOPEN MQOO_BROWSE, MQOO_INPUT_* o MQOO_SET requieren una instancia local de la cola de clúster para que MQOPEN funcione.

Las opciones MQOPEN MQOO_OUTPUT, MQOO_BIND_* o MQOO_INQUIRE no requieren que una instancia local del clúster funcione.

Nombre del gestor de colas resuelto

Cuando un nombre de gestor de colas se resuelve en tiempo de MQOPEN, el nombre resuelto se devuelve a la aplicación. Si la aplicación intenta utilizar este nombre en una llamada MQOPEN posterior, puede que se encuentre con que carece de autorización para acceder al nombre.

MQPUT, MQPUT1 y clústeres

Si se especifica MQOO_BIND_NOT_FIXED en una MQOPEN, las rutinas de gestión de carga de trabajo eligen qué destino MQPUT o MQPUT1 seleccionan.

Si se especifica MQOO_BIND_NOT_FIXED en una llamada MQOPEN, cada llamada MQPUT posterior invocará la rutina de gestión de carga de trabajo para determinar a qué gestor de colas se va a enviar el mensaje. El destino y la ruta que se van a tomar se seleccionan mensaje a mensaje. El destino y la ruta pueden cambiar después de haberse colocado el mensaje si cambian las condiciones de red. La llamada MQPUT1 siempre funciona como si MQOO_BIND_NOT_FIXED estuviera en vigor, es decir, invoca siempre la rutina de gestión de carga de trabajo.

Cuando la rutina de gestión de carga de trabajo ha seleccionado un gestor de colas, el gestor de colas local completa la operación de colocación. El mensaje se puede colocar en colas diferentes:

1. Si el destino es la instancia local de la cola, el mensaje se coloca en la cola local.
2. Si el destino es un gestor de colas en un clúster, el mensaje se coloca en una cola de transmisión de clúster.
3. Si el destino es un gestor de colas fuera de un clúster, el mensaje se coloca en una cola de transmisión con el mismo nombre que el gestor de colas de destino.

Si se especifica MQOO_BIND_ON_OPEN en la llamada MQOPEN, las llamadas MQPUT no invocarán la rutina de gestión de carga de trabajo, porque el destino y la ruta ya se han seleccionado.

MQINQ y clústeres

La cola de clúster que se consulta depende de las opciones que se combinan con MQOO_INQUIRE.

Para poder realizar consultas en una cola, ábrala utilizando la llamada MQOPEN y especifique MQOO_INQUIRE.

Para realizar consultas en una cola de clúster, utilice la llamada MQOPEN y combine otras opciones con MQOO_INQUIRE. Los atributos que pueden consultarse dependen de si hay una instancia local de la cola de clúster y de cómo se abre la cola:

- La combinación de MQOO_BROWSE, MQOO_INPUT_* o MQOO_SET con MQOO_INQUIRE requiere una instancia local de la cola de clúster para que la apertura tenga éxito. En este caso, puede consultar todos los atributos que son válidos para las colas locales.
- Si se combina MQOO_OUTPUT con MQOO_INQUIRE y no se especifica ninguna de las opciones anteriores, la instancia puede ser:
  - La instancia en el gestor de colas local, si hay una. En este caso, puede consultar todos los atributos que son válidos para las colas locales.
  - Una instancia en otro lugar del clúster, si no hay ninguna instancia de gestor de colas local. En este caso, solo pueden consultarse los siguientes atributos. El atributo QType tiene el valor MQQT_CLUSTER en este caso.

    - DefBind

    - DefPersistence

- DefPriority
- InhibitPut
- QDesc
- QName
- QType

Para consultar el atributo DefBind de una cola de clúster, utilice la llamada MQINQ con el selector MQIA_DEF_BIND. El valor devuelto es MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED o MQBND_BIND_ON_GROUP.Se debe especificar MQBND_BIND_ON_OPEN o MQBND_BIND_ON_GROUP cuando se utilizan grupos con clústeres.

Para consultar los atributos CLUSTER y CLUSNL de la instancia local de una cola, utilice la llamada MQINQ con el selector MQCA_CLUSTER_NAME o el selector MQCA_CLUSTER_NAMELIST.

Nota: Si abre una cola de clúster sin solucionar la cola con la que se ha enlazado MQOPEN, las llamadas MQINQ sucesivas pueden consultar distintas instancias de la cola de clúster.

Conceptos relacionados

"Opción MQOPEN para cola de clúster" en la página 768
El enlace utilizado para el manejador de cola se toma del atributo de cola **DefBind**, que puede tomar el valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, o MQBND_BIND_ON_GROUP.

MQSET y clústeres

La opción de MQOPEN MQOO_SET requiere que haya una instancia local de una cola de clúster para que MQSET funcione.

No puede utilizar la llamada MQSET para establecer los atributos de una cola en otro lugar del clúster.

Se puede abrir un alias local o una cola remota definida con el atributo de clúster y utilizar la llamada MQSET. Se pueden establecer los atributos del alias local o de la cola remota. No importa si la cola de destino es una cola de clúster definida en un gestor de colas distinto.

Códigos de retorno

Códigos de retorno específicos de un clúster

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Se emite una llamada MQOPEN, MQPUT o MQPUT1 para abrir una cola de clúster o colocar un mensaje en ella. La salida de carga de trabajo de clúster, definida por el atributo ClusterWorkloadExit de un gestor de colas, falla inesperadamente o no responde a tiempo.

Se escribe un mensaje en el registro del sistema en IBM MQ for z/OS lo que proporciona más información sobre este error.

Las siguientes llamadas MQOPEN, MQPUT y MQPUT1 de este descriptor de cola se procesarán como si el atributo ClusterWorkloadExit estuviera en blanco.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

En z/OS, no se puede cargar la salida de la carga de trabajo de clúster.

Se escribe un mensaje en el registro del sistema y el proceso continúa como si el atributo ClusterWorkloadExit estuviera en blanco.

Multi En Multiplatforms, se emite una llamada MQCONN o MQCONNX para conectarse a un gestor de colas. La llamada falla porque no se puede cargar la salida de carga de trabajo de clúster definida en el atributo de gestor de colas ClusterWorkloadExit.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Se emite una llamada MQOPEN con las opciones MQOO_OUTPUT y MQOO_BIND_ON_OPEN en vigor para una cola de clúster. Todas las instancias de la cola en el clúster tienen actualmente inhibida la colocación al tener el atributo InhibitPut establecido a MQQA_PUT_INHIBITED. Puesto que no hay instancias de cola disponibles para recibir mensajes, la llamada MQOPEN falla.

Este código de razón solo se produce cuando las dos sentencias siguientes son verdaderas:

- No hay ninguna instancia local de la cola. Si hay una instancia local, la llamada MQOPEN se realiza correctamente, incluso si la instancia local tiene inhibidas las colocaciones.
- No hay ninguna salida de carga de trabajo de clúster para la cola, o la hay, pero no elige una instancia de cola. (Si la salida de carga de trabajo de clúster elige una instancia de cola, la llamada MQOPEN será correcta, aunque dicha instancia tenga inhibidas las colocaciones).

Si se especifica la opción MQOO_BIND_NOT_FIXED en la llamada MQOPEN, esta puede ser satisfactoria incluso si todas las colas del clúster tienen inhibidas las colocaciones. Sin embargo, una llamada MQPUT posterior podría fallar si todas las colas siguen teniendo inhibidas las colocaciones en el momento de efectuarse dicha llamada.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Se emite una llamada MQOPEN, MQPUT o MQPUT1 para abrir una cola de clúster o colocar un mensaje en ella. La definición de cola no se puede resolver correctamente porque se necesita una respuesta del gestor de colas de repositorio completo, pero no hay ninguno disponible.
2. Se emite una llamada MQOPEN, MQPUT, MQPUT1 o MQSUB para un objeto de tema que especifica PUBSCOPE (ALL) o SUBSCOPE (ALL). La definición de tema de clúster no se puede resolver correctamente porque se necesita una respuesta del gestor de colas de repositorio completo, pero no hay ninguna disponible.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Se emite una llamada MQOPEN, MQPUT o MQPUT1 para una cola de clúster. Se produce un error al intentar usar un recurso necesario para agrupar en clúster.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Se emite una llamada MQPUT MQPUT1 para colocar un mensaje en una cola de clúster. En el momento de la llamada, ya no hay ninguna instancia de la cola en el clúster. MQPUT falla y el mensaje no se envía.

El error se puede producir si se especifica MQOO_BIND_NOT_FIXED en la llamada MQOPEN que abre la cola, o se usa MQPUT1 para colocar el mensaje.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Se emite una llamada MQOPEN, MQPUT o MQPUT1 para abrir o colocar un mensaje en en una cola de clúster. La salida de carga de trabajo de clúster rechaza la llamada.

z/OS Utilización y escritura de aplicaciones en IBM MQ for z/OS

Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

Esta información explica los recursos de IBM MQ disponibles para los programas que se ejecutan en cada uno de los entornos soportados. Además,

- Para obtener más información sobre cómo utilizar IBM MQ-CICS bridge, consulte Utilización de IBM MQ con CICS .
- Para obtener más información sobre cómo utilizar IMS y el puente IMS, consulte “IMS y las aplicaciones puente IMS en IBM MQ for z/OS” en la página 72.

Utilice los enlaces siguientes para descubrir más sobre el uso y la escritura de aplicaciones en IBM MQ for z/OS:

- “Funciones de IBM MQ for z/OS dependientes del entorno” en la página 912
- “Servicio de depuración, soporte de punto de sincronización y soporte de recuperación” en la página 913
- “Interfaz IBM MQ for z/OS con el entorno de aplicación” en la página 914

- “Escritura de aplicaciones de z/OS UNIX System Services” en la página 915
- “Programación de aplicaciones con colas compartidas” en la página 919

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Confirmación y restitución de unidades de trabajo” en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“IMS y las aplicaciones puente IMS en IBM MQ for z/OS” en la página 72
Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

z/OS *Funciones de IBM MQ for z/OS dependientes del entorno*

Utilice esta información al trabajar con funciones de IBM MQ for z/OS.

Estas son las diferencias principales que se deben tener en cuenta entre las funciones de IBM MQ en los entornos en los que se ejecuta IBM MQ for z/OS:

- IBM MQ for z/OS proporciona los supervisores desencadenantes siguientes:
  - CKTI para su uso en el entorno de CICS
  - CSQQTRMN para su uso en el entorno de IMS

  Debe escribir su propio módulo para iniciar aplicaciones en otros entornos.
- El proceso de puntos de sincronización utilizando la confirmación en dos fases está soportado en los entornos de CICS e IMS. También está soportado en el entorno de proceso por lotes de z/OS utilizando la gestión de transacciones y los servicios de gestor de recursos recuperables (RRS). La confirmación en una sola fase está soportada en el entorno z/OS por el propio IBM MQ.
- Para los entornos de proceso por lotes y de IMS, MQI proporciona llamadas para conectar programas con un gestor de colas y para desconectar programas de él. Los programas se pueden conectar a más de un gestor de colas.
- Un sistema CICS puede conectar con un solo gestor de colas. Esto puede ocurrir cuando se inicia CICS si el nombre de subsistema está definido en el trabajo de arranque del sistema CICS. Las llamadas de conexión y desconexión de MQI están permitidas, pero no tienen ningún efecto en el entorno CICS.
- Una salida cruzada de API permite que un programa intervenga en el proceso de todas las llamadas MQI. Esta salida sólo está disponible en el entorno de CICS.

- En los sistemas multiprocesador de CICS se obtiene alguna ventaja de rendimiento porque las llamadas MQI se pueden ejecutar bajo varios TCB de z/OS. Para obtener más información, consulte la publicación Planificación en z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Estas características se resumen en la Tabla 130 en la página 913.

Tabla 130. Características del entorno de z/OS

	CICS	**IMS**	**Batch/TSO**
Supervisor desencadenante proporcionado	Sí	Sí	No
Confirmación en dos fases	Sí	Sí	Sí
confirmación en una sola fase	Sí	No	Sí
Llamadas MQI de conexión/desconexión	Permitidas	Sí	Sí
salida cruzada de API	Sí	No	No

Nota: La confirmación en dos fases está soportada en el entorno Batch/TSO utilizando RRS.

z/OS *Servicio de depuración, soporte de punto de sincronización y soporte de recuperación*

Aquí puede obtener información sobre los recursos de depuración de programas, el soporte de punto de sincronización y el soporte de recuperación.

Servicios de depuración de programas

IBM MQ for z/OS proporciona un servicio de rastreo que puede utilizar para depurar los programas en todos los entornos.

Además, en el entorno de CICS puede utilizar:

- El recurso de diagnóstico de ejecución de CICS (CEDF)
- La transacción de control de rastreo de CICS (CETR)
- La salida cruzada de API de IBM MQ for z/OS

En la plataforma z/OS, puede utilizar cualquier herramienta de depuración interactiva disponible admitida por el lenguaje de programación que está utilizando.

Soporte de punto de sincronización

La sincronización del inicio y el final de las unidades de trabajo es necesaria en un entorno de proceso de transacciones para que el proceso de transacciones se pueda utilizar de forma segura.

Esto está completamente soportado por IBM MQ for z/OS en los entornos CICS y IMS. El soporte completo significa la cooperación entre gestores de recursos para que las unidades de trabajo se puedan confirmar o restituir al unísono, bajo el control de CICS o IMS. Algunos ejemplos de gestores de recursos son Db2, CICS File Control, IMS y IBM MQ for z/OS.

Las aplicaciones de proceso por lotes de z/OS pueden utilizar llamadas de IBM MQ for z/OS para proporcionar un servicio de confirmación de una sola fase. Esto significa que se puede confirmar o restituir un conjunto definido por la aplicación de operaciones de cola sin hacer referencia a otros gestores de recursos.

También se admite la confirmación en dos fases en el entorno de proceso por lotes de z/OS mediante gestión de transacciones y servicios de gestor de recurso recuperable (RRS). Para obtener más información, consulte Puntos de sincronización en aplicaciones por lotes z/OS.

Soporte de recuperación

Si la conexión entre un gestor de colas y un sistema CICS o IMS se rompe durante una transacción, es posible que algunas unidades de trabajo no se restituyan correctamente.

Sin embargo, estas unidades de trabajo se resuelven mediante el gestor de colas (bajo el control del gestor de puntos de sincronización) cuando se vuelve a establecer su conexión con el sistema CICS o IMS.

z/OS *Interfaz IBM MQ for z/OS con el entorno de aplicación*

Para permitir que las aplicaciones que se ejecutan en entornos diferentes envíen y reciban mensajes a través de una red de colocación en cola de mensajes, IBM MQ for z/OS proporciona un *adaptador* para cada uno de los entornos soportados.

Estos adaptadores son la interfaz entre los programas de aplicación y los subsistemas de IBM MQ for z/OS. Perpermiten que los programas utilicen MQI.

z/OS *El adaptador por lotes*

Utilice esta información para obtener información sobre el adaptador por lotes y el protocolo de confirmación al que da soporte. por él.

El *adaptador por lotes* proporciona acceso a recursos de IBM MQ for z/OS para los programas que se ejecutan en:

- Modalidad (TCB) de tarea
- Problema o estado de supervisor
- Modalidad de control de espacio de direcciones primaria

Los programas no deben estar en la modalidad entre memorias.

Las conexiones entre programas de aplicación y IBM MQ for z/OS están en el nivel de tarea. El adaptador proporciona una sola hebra de conexión de un bloque de control de tareas de aplicación (TCB) a IBM MQ for z/OS.

El adaptador es compatible con el protocolo de confirmación en una sola fase para los cambios realizados en recursos que son propiedad de IBM MQ for z/OS; no da soporte a los protocolos de confirmación en varias fases.

z/OS *El adaptador de proceso por lotes RRS*

Utilice esta sección para obtener información sobre el adaptador de procesos por lotes RRS y los dos adaptadores por lotes RRS que proporciona IBM MQ.

La gestión de transacciones y el adaptador RRS (Recoverable Resource Manager Services):

- Utiliza z/OS RRS para control de confirmación.
- Da soporte a conexiones simultáneas con varios subsistemas IBM MQ que se ejecutan en una sola instancia de z/OS de una única tarea.
- Proporciona un control coordinado de confirmación en todo el sistema z/OS, utilizando z/OS RRS para los recursos recuperables a los que se accede mediante los gestores recuperables compatibles con z/OS RRS para :
  - Aplicaciones que se conectan a IBM MQ utilizando el adaptador por lotes RRS.
  - Procedimientos almacenados de Db2 que se ejecutan en un espacio de direcciones de procedimientos almacenados de Db2 gestionado por el gestor de carga de trabajo (WLM) en z/OS.
- Da soporte a la función de conmutar una hebra de IBM MQ por lote entre TCB.

IBM MQ for z/OS proporciona dos adaptadores RRS por lotes:

CSQBRSTB
Este adaptador requiere que cambie cualquier sentencia MQCMIT por SRRCMIT y cualquier sentencia MQBACK por SRRBACK en su aplicación IBM MQ. Si codifica MQCMIT o MQBACK en una aplicación enlazada con CSQBRSTB, recibirá MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
Este adaptador permite que su aplicación IBM MQ utilice MQCMIT y MQBACK o SRRCMIT y SRRBACK.

Nota: CSQBRSTB y CSQBRRSI se incluyen con los atributos de enlace AMODE(31) RMODE(ANY). Si su aplicación carga cualquier apéndice por debajo de la línea de los 16 MB, en primer lugar, vuelva a enlazar el apéndice con RMODE(24).

Migración

Puede migrar las aplicaciones IBM MQ por lotes/TSO para que utilicen la coordinación de RRS con algunos cambios o ninguno.

Si edita enlaces en su aplicación IBM MQ con el adaptador CSQBRRSI, MQCMIT y MQBACK crean un punto de sincronismo de su unidad de trabajo entre IBM MQ y todos los gestores de colas habilitados para RRS. Si edita enlaces en su aplicación IBM MQ con el adaptador CSQBRSTB, cambie MQCMIT por SRRCMIT y MQBACK por SRRBACK. Este último método es el preferido, ya que indica claramente que el punto de sincronismo no está restringido únicamente a los recursos de IBM MQ.

z/OS *El adaptador IMS*
Si utiliza el adaptador IMS desde un sistema IBM MQ for z/OS, asegúrese de que IMS puede obtener almacenamiento suficiente para dar cabida a mensajes de hasta 100 MB de longitud.

Nota para los usuarios

El *adaptador IMS* proporciona acceso a los recursos de IBM MQ for z/OS para:

- Programas de proceso de mensajes en línea (MPP)
- Programas de vía de acceso rápida interactiva (IFP)
- Programas de proceso de mensajes por lotes (BMP)

Para utilizar estos recursos, los programas deben estar en ejecución en la modalidad de tarea (TCB) y estado de problema; no deben estar en la modalidad entre memorias ni en la modalidad de registro de acceso.

El adaptador proporciona una hebra de conexión desde un bloque de control de tareas de aplicación (TCB) a IBM MQ. El adaptador es compatible con el protocolo de confirmación en dos fases para los cambios realizados en recursos que son propiedad de IBM MQ for z/OS, con IMS actuando como coordinador de puntos de sincronización.

El adaptador también proporciona un programa supervisor desencadenante que puede iniciar programas automáticamente cuando se cumplen determinadas condiciones de desencadenante en una cola. Para obtener más información, consulte “Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886.

Si está escribiendo programas de DL/I de proceso por lotes, siga las instrucciones que se proporcionan en este tema para programas de proceso por lotes de z/OS.

z/OS *Escritura de aplicaciones de z/OS UNIX System Services*

El adaptador por lotes da soporte a conexiones de gestor de colas desde espacios de direcciones por lotes y TSO.

Para un espacio de direcciones por lotes, el adaptador da soporte a conexiones de varios TCB dentro de ese espacio de direcciones como se indica a continuación:

- Cada TCB se puede conectar a varios gestores de colas utilizando la llamada MQCONN o MQCONNX (pero un TCB sólo puede tener una instancia de una conexión con un gestor de colas determinado en cada momento).
- Varios TCB pueden conectarse al mismo gestor de colas (pero el manejador de gestor de colas devuelto en cualquier llamada MQCONN o MQCONNX está enlazado con el TCB emisor y no puede ser utilizado por ningún otro TCB).

z/OS UNIX System Services da soporte a dos tipos de llamada pthread_create:

1. Las hebras intensas, se ejecutan una vez en cada TCB, que se CONECTA y DESCONECTA (ATTACH/DETACH) al inicio y fin de la hebra, por parte de z/OS.
2. Las hebras de tamaño medio se ejecutan una vez para cada TCB, pero el TCB puede ser uno del grupo de los TCB de larga ejecución. La aplicación debe realizar toda la limpieza necesaria de la aplicación, ya que, si está conectada a un servidor, la terminación de hebra predeterminada que puede proporcionar la terminación del servidor en la tarea (TCB), **no** siempre se lleva a cabo.

Las hebras ligeras no se admiten. (Si una aplicación crea hebras permanentes que envían sus propias solicitudes de trabajo, la ***aplicación*** es responsable de la limpieza de cualquier recurso antes de iniciar la siguiente solicitud de trabajo.)

IBM MQ for z/OS da soporte a las hebras de z/OS UNIX System Services utilizando el adaptador por lotes de la forma siguiente:

1. Las hebras de gran tamaño se admiten completamente como conexiones por lotes. Cada hebra se ejecuta en su propio TCB, que se conecta y desconecta al inicio y final de la hebra. Si la hebra termina antes de emitir una llamada MQDISC, IBM MQ for z/OS realiza su limpieza estándar de tareas, que incluye la confirmación de cualquier unidad de trabajo pendiente, si la hebra ha terminado con normalidad, o la restituye si la hebra no ha finalizado normalmente.
2. Las hebras de tamaño medio están totalmente soportadas, pero si el TCB va a ser reutilizado por otra hebra, la aplicación debe asegurarse de que se emita una llamada MQDISC, precedida por MQCMIT o MQBACK, antes de que se inicie la siguiente hebra. Esto implica que si la aplicación ha establecido un manejador de interrupción de programa y la aplicación termina de forma anómala, el Manejador de interrupciones debe emitir llamadas MQCMIT y MQDISC antes de volver a utilizar el TCB para otra hebra.

Nota: Los modelos de hebras **no** admiten al acceso a recursos comunes de IBM MQ desde varias hebras.

z/OS *La salida cruzada de API para z/OS*

En este tema se incluye información sobre la interfaz de programación dependiente del producto.

Una salida es un punto del código proporcionado por IBM en el que puede ejecutar su propio código. IBM MQ for z/OS proporciona una *salida cruzada de API* que puede utilizar para interceptar llamadas a MQI y supervisar o modificar la función de las llamadas MQI. En esta sección se describe cómo utilizar la salida cruzada de API y el programa de salida de ejemplo que se proporciona con IBM MQ for z/OS.

Esta sección solo es aplicable para los usuarios de CICS TS V3.1 y anterior. Los usuarios de CICS TS V3.2 y posterior deben consultar la sección Integración de CICS con IBM MQ en la documentación del producto CICS.

Nota

La salida cruzada de API solo la invoca el adaptador CICS de IBM MQ for z/OS. El programa de salida se ejecuta en el espacio de direcciones de CICS.

z/OS *Cómo escribir su propio programa de salida*

Puede utilizar el programa de salida cruzada de API (CSQCAPX) proporcionado con IBM MQ for z/OS como infraestructura para su propio programa.

Este tema se describe en el apartado “Ejemplo de programa de salida cruzada de API, CSQCAPX” en la página 917.

Al escribir un programa de salida, para buscar el nombre de una llamada MQI emitida por una aplicación, examine el campo *ExitCommand* de la estructura de MQXP. Para buscar el número de parámetros de la llamada, examine el campo *ExitParmCount*. Puede utilizar el campo *ExitUserArea* de 16 bytes para almacenar la dirección de cualquier almacenamiento dinámico que obtenga la aplicación. Este campo se retiene entre invocaciones de la salida y tiene el mismo tiempo de vida que una tarea CICS.

Si utiliza CICS Transaction Server V3.2, debe escribir su programa de salida para que sea de hebra segura y declarar que el programa de salida es de hebra segura. Si utiliza releases anteriores de CICS, se recomienda que también escriba y declare sus programas de salida como de hebra segura para que estén listos para la migración a CICS Transaction Server V3.2.

Su programa de salida puede suprimir la ejecución de una llamada MQI devolviendo MQXCC_SUPPRESS_FUNCTION o MQXCC_SKIP_FUNCTION en el campo *ExitResponse*. Para permitir que se pueda ejecutar la llamada (y que se vuelva a invocar el programa de salida después de que finalice la llamada), el programa de salida debe devolver MQXCC_OK.

Cuando se invoca después de una llamada MQI, un programa de salida puede inspeccionar y modificar los códigos de terminación y de razón establecidos por la llamada.

Notas de uso

A continuación se muestran algunos puntos generales a tener en cuenta al escribir programas de salida:

- Por motivos de rendimiento, escriba el programa en lenguaje ensamblador. Si lo escribe en cualquier de los demás lenguajes admitidos por IBM MQ for z/OS, debe proporcionar su propio archivo de definición de datos.
- Edite el enlace de su programa como AMODE(31) y RMODE(ANY).
- Para definir el bloque de parámetros de salida para el programa, utilice la macro de lenguaje ensamblador CMQXPA.
- Especifique CONCURRENCY(THREADSAFE) al definir su programa de salida y cualquier programa al que llame su programa de salida.
- Si utiliza CICS Transaction Server para la función de protección de almacenamiento de z/OS, el programa debe ejecutarse en la clave de ejecución de CICS. Es decir, debe especificar EXECKEY(CICS) al definir tanto el programa de salida como los programas a los que este pase el control. Para obtener información sobre programas de salida CICS y el recurso de protección de almacenamiento de CICS, consulte la *Guía de personalización de CICS*.
- El programa puede utilizar todas las API (por ejemplo, IMS, Db2 y CICS) que pueda utilizar un programa de salida de usuario relacionado con una tarea de CICS. También puede utilizar cualquiera de las llamadas MQI, excepto MQCONN, MQCONNX y MQDISC. No obstante, las llamadas MQI dentro del programa de salida no invocarán el programa de salida una segunda vez.
- El programa puede emitir los mandatos EXEC CICS SYNCPOINT o EXEC CICS SYNCPOINT ROLLBACK. No obstante, estos mandatos confirman o retrotraen ***todas*** las actualizaciones realizadas por la tarea hasta el punto en que se utilizó la salida y, por lo tanto, no se recomienda su uso.
- El programa debe terminar emitiendo un mandato EXEC CICS RETURN. No debe transferir el control con un mandato XCTL.
- Las salidas se escriben como extensiones al código de IBM MQ for z/OS. Asegúrese de que la salida no interrumpe ninguno de los programas o transacciones de IBM MQ for z/OS que utilicen la MQI. Estos se indican normalmente con el prefijo CSQ o CK.
- Si se define CSQCAPX en CICS, el sistema CICS intenta cargar el programa de salida cuando CICS se conecta a IBM MQ for z/OS. Si este intento tiene éxito, se envía el mensaje CSQC301I al panel CKQC o a la consola del sistema. Si la carga no se realiza correctamente (por ejemplo, si el módulo de carga no existe en ninguna de las bibliotecas de la concatenación DFHRPL), se envía el mensaje CSQC315 al panel CKQC o a la consola del sistema.
- Debido a que los parámetros del área de comunicación son direcciones, el programa de salida debe estar definido como local para el sistema CICS (es decir, no como un programa remoto).

z/OS *Ejemplo de programa de salida cruzada de API, CSQCAPX*

El programa de salida de ejemplo se proporciona en ensamblador. El archivo del código fuente (CSQCAPX) se proporciona en la biblioteca **thlqual**.SCSQASMS (donde **thlqual** es el calificador de alto nivel utilizado por la instalación). Este archivo fuente incluye pseudocódigo que describe la lógica del programa.

El programa de ejemplo contiene código de inicialización y un diseño que puede utilizar para escribir sus propios programas de salida.

En el ejemplo se muestra cómo:

- Configurar el bloque de parámetros de salida.
- Direccionar los bloques de parámetros de llamada y salida.
- Determinar para qué llamada MQI se invoca la salida.
- Determinar si la salida se está invocando antes o después del procesamiento de la llamada MQI.
- Colocar un mensaje en una cola de almacenamiento temporal de CICS.
- Utilizar la macro DFHEIENT para la adquisición de almacenamiento dinámico a fin de conservar la reentrancia.
- Utilizar DFHEIBLK para el bloque de control de la interfaz EXEC de CICS.
- Capturar condiciones de error.
- Devolver el control al llamante.

Diseño del programa de salida de ejemplo

El programa de salida de ejemplo escribe mensajes en una cola de almacenamiento temporal CICS (CSQ1EXIT) para mostrar el funcionamiento de la salida.

Los mensajes muestran tanto si la salida se invoca antes como después de la llamada MQI. Si la salida se invoca después de la llamada, el mensaje contendrá los códigos de terminación y de razón devueltos por la llamada. El ejemplo utiliza constantes de la macro CMQXPA para comprobar el tipo de entrada (es decir, anterior o posterior a la llamada).

El ejemplo no realiza ninguna función de supervisión, sino que simplemente coloca los mensajes con indicación de fecha y hora en una cola de CICS que indica el tipo de llamada que está procesando. Esto proporciona una indicación del rendimiento de la MQI, así como el funcionamiento correcto del programa de salida.

Nota: El programa de salida de ejemplo emite seis llamadas EXEC de CICS para cada llamada MQI que se realiza mientras se está ejecutando el programa. Si utiliza este programa de salida, se degrada el rendimiento de IBM MQ for z/OS.

z/OS *Preparación y uso de la salida cruzada de API*

La salida de ejemplo solo se proporciona en formato de código fuente.

Para utilizar la salida de ejemplo, o un programa de salida que haya escrito, cree una biblioteca de carga, tal como lo haría para cualquier otro programa CICS, tal como se describe en “Creación de aplicaciones CICS en z/OS” en la página 1053.

- En CICS Transaction Server para z/OS y CICS para MVS/ESA, cuando se actualiza el conjunto de datos de definición de sistema CICS (CICS System Definition, CSD), las definiciones necesarias están en el miembro **thlqual**.SCSQPROC(CSQ4B100).

  Nota: Las definiciones utilizan el sufijo MQ. Si este sufijo ya se utiliza en la empresa, hay que cambiarlo antes de la etapa de ensamblaje.

Si utiliza las definiciones de programa predeterminadas de CICS proporcionadas, el programa de salida CSQCAPX se instala en un estado ***inhabilitado*** . Esto se debe a que el uso del programa de salida puede penalizar significativamente el rendimiento.

Para activar temporalmente la salida cruzada de API:

1. Emita el mandato **CEMT S PROGRAM(CSQCAPX) ENABLED** desde el terminal maestro de CICS .
2. Ejecute la transacción CKQC y utilice la opción 3 del menú desplegable Conexión para modificar el estado de la salida cruzada de API a **Habilitada**.

Si desea ejecutar IBM MQ for z/OS con la salida cruzada de API permanentemente habilitada, en CICS Transaction Server para z/OS y CICS para MVS/ESA, haga una de las cosas siguientes:

- Modifique la definición de CSQCAPX en el miembro CSQ4B100, cambiando STATUS(DISABLED) a STATUS(ENABLED). Puede actualizar la definición de CSD de CICS utilizando el programa por lotes DFHCSDUP proporcionado por CICS.
- Modifique la definición de CSQCAPX del grupo CSQCAT1 cambiando el estado de DISABLED a ENABLED.

En ambos casos, hay que reinstalar el grupo. Puede hacerlo mediante un arranque en frío del sistema CICS o utilizando la transacción CEDA CICS para volver a instalar el grupo mientras CICS se está ejecutando.

Nota: El uso de CEDA podría provocar un error si alguna de las entradas del grupo se está usando en ese momento.

Fin de la información de interfaz de programación dependiente del producto.

z/OS *Programación de aplicaciones con colas compartidas*

Este tema proporciona información sobre algunos de los factores que debe tener en cuenta al diseñar aplicaciones nuevas que utilicen colas compartidas, y al migrar aplicaciones existentes al entorno de colas compartidas.

z/OS *Serialización de las aplicaciones*

Es posible que algunos tipos de aplicaciones tengan que asegurarse de que los mensajes se recuperan de una cola exactamente en el mismo orden en el que llegaron a la cola.

Por ejemplo, si se está utilizando IBM MQ para duplicar las actualizaciones de base de datos en un sistema remoto, debe procesarse un mensaje que describa la actualización en un registro después del mensaje que describe la inserción de ese registro. En un entorno de gestión de colas local, esto se consigue a menudo cuando la aplicación que obtiene los mensajes abre la cola con la opción MQOO_INPUT_EXCLUSIVE, lo que impide que cualquier otra aplicación que obtenga mensaje procese la cola al mismo tiempo.

IBM MQ permite a las aplicaciones abrir colas compartidas de manera exclusiva de la misma forma. Sin embargo, si la aplicación está trabajando desde una partición de una cola (por ejemplo, si todas las actualizaciones de base de datos están en la misma cola, pero las de la tabla A tienen un identificador de correlación A y las de la tabla B tienen un identificador de correlación B), y las aplicaciones desean obtener mensajes para las actualizaciones de la tabla A y las actualizaciones de tabla B simultáneamente, el simple mecanismo de abrir la cola de manera exclusiva no es posible.

Si este tipo de aplicación va a aprovechar la alta disponibilidad de las colas compartidas, puede decidir que otra instancia de la aplicación que accede a las mismas colas compartidas, que se ejecutan en un gestor de colas secundario, debe tomar el control si el gestor de colas primario o la aplicación de obtención primaria falla.

Si el gestor de colas primario falla, ocurren dos cosas:

- La recuperación de igual de cola compartida garantiza que todas las actualizaciones incompletas de la aplicación primaria se completen o restituyan.
- La aplicación secundaria toma el control del proceso de la cola.

Es posible que la aplicación secundaria se inicie antes de que se hayan resuelto todas las unidades de trabajo incompletas, lo que puede dar como resultado que la aplicación secundaria recupere los mensajes fuera de secuencia. Para solucionar este tipo de problema, la aplicación puede optar por ser una *aplicación serializada*.

Una aplicación serializada utiliza la llamada MQCONNX para conectarse al gestor de colas, especificando una etiqueta de conexión cuando se conecta que es exclusiva de dicha aplicación. Las unidades de trabajo ejecutadas por la aplicación se marcan con la etiqueta de conexión. IBM MQ garantiza que las unidades de trabajo dentro del grupo de compartición de colas con el mismo código de conexión se serializan (de acuerdo con las opciones de serialización de la llamada MQCONNX).

Esto significa que, si la aplicación primaria utiliza la llamada MQCONNX con una etiqueta de conexión `Database shadow retriever` y la aplicación de toma de control secundaria intenta utilizar la llamada MQCONNX con una etiqueta de conexión idéntica, la aplicación secundaria no se puede conectar al segundo IBM MQ hasta que se hayan completado todas las unidades de trabajo primarias pendientes, en este caso mediante la recuperación de igual.

Se recomienda utilizar la técnica de aplicación serializada para las aplicaciones que dependen de la secuencia exacta de mensajes en una cola. En concreto:

- Las aplicaciones que no deben reiniciarse después de un error de la aplicación o el gestor de colas hasta que todas las operaciones de confirmación y restitución de la ejecución anterior de la aplicación hayan finalizado.

  En este caso, la técnica de aplicación serializada solo es aplicable si la aplicación funciona en el punto de sincronización.
- Las aplicaciones que no deben iniciarse mientras se esté ejecutando otra instancia de la misma aplicación.

  En este caso, la técnica de aplicación serializada solo es necesaria si la aplicación no puede abrir la cola para la entrada exclusiva.

Nota: IBM MQ solo garantiza la conservación de la secuencia de mensajes cuando se cumplen determinados criterios. Estos se especifican en la descripción de MQGET.

z/OS *Aplicaciones que no son adecuadas para su uso con colas compartidas*

Algunas características de IBM MQ no están soportadas cuando se utilizan colas compartidas, por lo que las aplicaciones que utilizan estas características no son adecuadas para el entorno de colas compartidas.

Tenga en cuenta lo siguiente al diseñar aplicaciones de cola compartida:

- La indexación de colas está limitada para las colas compartidas. Si desea utilizar el identificador de mensaje o identificador de correlación para seleccionar el mensaje que desea obtener de la cola, la cola se debe indexar con el valor correcto. Si selecciona mensajes sólo mediante el identificador de mensaje, la cola necesita el tipo de índice MQIT_MSG_ID (aunque también puede utilizar MQIT_NONE). Si selecciona mensajes sólo mediante el identificador de correlación, la cola debe tener el tipo de índice MQIT_CORREL_ID.
- No puede utilizar colas dinámicas temporales como colas compartidas. Pero puede utilizar las colas dinámicas permanentes. Los modelos para las colas dinámicas compartidas tienen un DEFTYPE igual a SHAREDYN (dinámicas compartidas), aunque se crean y destruyen de la misma forma que las colas PERMDYN (dinámicas permanentes).

z/OS *Decidir si se deben compartir las colas no de aplicación*

Utilice esta información cuando se plantee la posibilidad de compartir colas de no aplicación.

Hay colas distintas de las colas de aplicaciones que es posible que quiera compartir:

Colas de inicio

Si define una cola de inicio compartida, no es necesario que tenga un supervisor desencadenante que se ejecute en cada gestor de colas del grupo de compartición de colas, siempre que haya, al menos, un supervisor desencadenante en ejecución. (También puede utilizar una cola de inicio compartida, incluso si hay un supervisor desencadenante que se ejecuta en cada gestor de colas del grupo de compartición de colas.)

Si ha compartido una cola de aplicación y utiliza el tipo de desencadenante EVERY (o un cola de aplicación FIRST con un pequeño intervalo de desencadenante, que se comporte como un tipo de desencadenante EVERY), su cola de inicio siempre debe ser una cola compartida. Para obtener más información sobre cuándo utilizar una cola de inicio compartida, consulte Tabla 131 en la página 921.

SYSTEM.* colas

Puede definir SYSTEM.ADMIN.* las colas utilizadas para contener mensajes de sucesos como colas compartidas. Esto puede ser útil para comprobar el equilibrio de carga si se produce una excepción. Cada mensaje de suceso creado por IBM MQ contiene un identificador de correlación que indica qué gestor de colas lo ha generado.

Debe definir SYSTEM.QSG.* colas utilizadas para canales compartidos y colas dentro del grupo como colas compartidas.

También puede cambiar las definiciones de SYSTEM.DEFAULT.LOCAL.QUEUE para que se compartan, o definir su propia definición de cola compartida predeterminada. Consulte Definición de objetos del sistema para IBM MQ for z/OS para obtener más información.

No puede definir ningún otro SYSTEM.* como colas compartidas.

z/OS *Migración de sus aplicaciones existentes a colas compartidas*

Los códigos de razón, el desencadenamiento y la llamada a la API MQINQ pueden funcionar de forma diferente en un entorno de cola compartida.

Consulte Migración de colas no compartidas a colas compartidas para obtener información sobre cómo migrar las colas existentes a colas compartidas.

Cuando migre las aplicaciones existentes, tenga en cuenta los aspectos siguientes, que podrían funcionar de una forma distinta en el entorno de cola compartida:

códigos de razón

Cuando migre las aplicaciones existentes para que utilicen colas compartidas, compruebe los nuevos códigos de razón que se pueden emitir.

Desencadenamiento

Si utiliza una cola de aplicación compartida, el desencadenante sólo funciona en mensajes confirmados (en una cola de aplicación no compartida, el desencadenante funciona en todos los mensajes).

Si utiliza desencadenante para iniciar aplicaciones, es posible que quiera usar una cola de inicio compartida. Tabla 131 en la página 921 describe lo que debe tener en cuenta al decidir el tipo de cola de inicio a utilizar.

Tabla 131. Cuándo utilizar una cola de inicio compartido

	Cola de aplicaciones no compartidas	**Cola de aplicaciones compartidas**
Cola de inicio no compartida	Según los releases anteriores.	Si utiliza un tipo de desencadenante FIRST o DEPTH, puede utilizar una cola de inicio que no sea compartida con una cola de aplicación compartida. Es posible que se generen mensajes de desencadenante adicionales, pero esta configuración es buena para desencadenar aplicaciones de larga ejecución (como CICS bridge) y proporciona alta disponibilidad. Para el tipo de desencadenante FIRST o DEPTH, un mensaje desencadenante activa una instancia de la aplicación en cada gestor de colas que ejecuta un supervisor desencadenante y que no tenga una cola de aplicación abierta para la entrada. Se genera un mensaje desencadenante para cada gestor de colas; si hay más de un supervisor desencadenante en ejecución en la cola de inicio local no compartida, en un gestor de colas determinado, competirán para procesar el mensaje.

Tabla 131. Cuándo utilizar una cola de inicio compartido (continuación)		
	Cola de aplicaciones no compartidas	**Cola de aplicaciones compartidas**
Cola de inicio compartida	No utilice una cola de inicio compartida con una cola de aplicación no compartida.	Para el tipo de desencadenante EVERY, cuando una aplicación coloca un mensaje en una cola de aplicación compartida, el gestor de colas de transferencia determina qué gestores de colas tienen interés en el desencadenante-cada suceso y envía una notificación a uno de esos gestores de colas. En el gestor de colas notificado, la acción resultante es generar un mensaje desencadenante para la cola de iniciación., **Nota:** Si tiene una cola de aplicación compartida con un tipo de desencadenante de EVERY, utilice una cola de inicio compartida o puede perder mensajes de desencadenante en determinadas circunstancias; por ejemplo, un gestor de colas que falle. Para el tipo de desencadenante FIRST o DEPTH, se genera un mensaje desencadenante por cada gestor de colas que tiene la cola de inicio indicada abierta para entrada. **Nota:** Para el tipo de desencadenante FIRST o DEPTH, si una instancia de supervisor desencadenante está ocupada, abre la posibilidad de que los supervisores de desencadenante menos ocupados procesen más de un mensaje desencadenante de la cola de inicio compartida. Por lo tanto, es posible que se inicien varias instancias de la aplicación de servidor contra un determinado gestor de colas. Tenga en cuenta que estas múltiples instancias se inician como resultado del proceso de varios mensajes desencadenantes. Por lo general, para el tipo de desencadenante FIRST o DEPTH, si una instancia de aplicación ya está sirviendo una cola de aplicación, el gestor de colas al que está conectada la aplicación no generará otro mensaje desencadenante.

MQINQ
Cuando utilice la llamada MQINQ para mostrar información sobre una cola compartida, los valores del número de llamadas MQOPEN que la cola abre para entrada y salida solo se relacionan con el gestor de colas que ha emitido la llamada. No se genera información sobre otros gestores de colas en el grupo de compartición de colas que tengan la cola abierta.

z/OS IMS y las aplicaciones puente IMS en IBM MQ for z/OS

Esta información le ayuda a escribir aplicaciones IMS utilizando IBM MQ.

- Para utilizar puntos de sincronización y llamadas MQI en aplicaciones IMS, consulte "Escritura de aplicaciones IMS utilizando IBM MQ" en la página 72.
- Para escribir aplicaciones que utilicen el puente IBM MQ - IMS, consulte "Escritura de aplicaciones puente IMS" en la página 76.

Consulte estos enlaces para obtener información adicional relativa a IMS y las aplicaciones puente IMS en IBM MQ for z/OS:

- “Escritura de aplicaciones IMS utilizando IBM MQ” en la página 72
- “Escritura de aplicaciones puente IMS” en la página 76

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 740
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 754
Para utilizar servicios de programación de IBM MQ, un programa debe tener una conexión con un gestor de colas. Utilice esta información para aprender a conectar y desconectar de un gestor de colas.

“Apertura y cierre de objetos” en la página 761
Esta información describe la apertura y cierre de objetos de IBM MQ.

“Colocación de mensajes en una cola” en la página 772
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 788
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 871
Los atributos son las propiedades que definen las características de un objeto de IBM MQ.

“Confirmación y restitución de unidades de trabajo” en la página 874
Esta información describe cómo confirmar y restituir cualquier operación get y put recuperable que se ha realizado en una unidad de trabajo.

“Inicio de aplicaciones de IBM MQ utilizando desencadenantes” en la página 886
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 907
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

“Utilización y escritura de aplicaciones en IBM MQ for z/OS” en la página 911
Las aplicaciones IBM MQ for z/OS pueden estar formadas por programas que se ejecutan en muchos entornos diferentes. Esto significa que pueden beneficiarse de los recursos disponibles en más de un entorno.

z/OS *Escritura de aplicaciones IMS utilizando IBM MQ*

Hay consideraciones adicionales cuando se utiliza IBM MQ en aplicaciones IMS. Estas incluyen las llamadas de API de MQ que se pueden utilizar y el mecanismo utilizado para el punto de sincronización.

Utilice los siguientes enlaces para obtener más información sobre la escritura de aplicaciones IMS en IBM MQ for z/OS:

- “Puntos de sincronismo en aplicaciones IMS” en la página 73
- “Llamadas MQI en aplicaciones IMS” en la página 73

Restricciones

Hay restricciones sobre qué llamadas de API de IBM MQ puede utilizar una aplicación que utiliza el adaptador IMS.

Las siguientes llamadas de API de IBM MQ no están soportadas en una aplicación que utiliza el adaptador IMS:

- MQCB
- MQCB_FUNCTION
- MQCTL

Conceptos relacionados

"Escritura de aplicaciones puente IMS" en la página 76

Este tema contiene información sobre cómo escribir aplicaciones para utilizar el puente IBM MQ - IMS.

z/OS *Puntos de sincronismo en aplicaciones IMS*

En una aplicación IMS, puede establecer un punto de sincronismo utilizando las llamadas de IMS, tales como GU (obtener único) para IOPCB y CHKP (punto de comprobación).

Para restituir todos los cambios desde el punto de comprobación anterior, puede utilizar la llamada ROLB (retrotraer) de IMS. Para obtener más información, consulte llamada ROLB en la documentación de IMS .

El gestor de colas participa en un protocolo de confirmación de dos fases. El gestor de puntos de sincronismo de IMS es el coordinador.

El adaptador IMS cierra todos los descriptores abiertos en un punto de sincronismo, excepto en un entorno por lotes o en un entorno BMP no controlado por mensajes. Esto es debido a que un usuario diferente podría iniciar la siguiente unidad de trabajo y cuando se emiten las llamadas MQCONN, MQCONNX y MQOPEN, se realiza la comprobación de seguridad de IBM MQ, y no cuando se emiten las llamadas MQPUT o MQGET.

No obstante, en un entorno WFI (Wait-for-Input) o en un entorno PWFI (Pseudo Wait-for-Input), IMS no notifica a IBM MQ que cierre los descriptores hasta que llegue el mensaje siguiente ni se devuelve un código de estado QC a la aplicación. Si la aplicación está esperando en la región de IMS y cualquiera de estos manejadores pertenece a las colas desencadenadas, no se llevará a cabo el desencadenamiento porque las colas están abiertas. Por este motivo, las aplicaciones que se ejecutan en un entorno WFI o PWFI deben emitir MQCLOSE de forma explícita para los descriptores de colas antes de ejecutar GU para IOPCB para el mensaje siguiente.

Si una aplicación IMS, ya sea BMP o MPP, emite la llamada MQDISC, se cierran las colas abiertas pero no se toma implícitamente un punto de sincronismo. Si la aplicación finaliza con normalidad, se cierra cualquiera cola abierta y se lleva a cabo una confirmación implícita. Si la aplicación finaliza de forma anómala, se cierra cualquiera cola abierta y se lleva a cabo una restitución implícita.

z/OS *Llamadas MQI en aplicaciones IMS*

Utilice esta información para obtener información sobre el uso de las llamadas MQI en aplicaciones de servidor y aplicaciones de consulta.

Esta sección cubre el uso de llamadas MQI en los tipos siguientes de aplicaciones IMS:

- "Aplicaciones de servidor" en la página 924
- "Aplicaciones de consulta" en la página 927

Aplicaciones de servidor

A continuación se muestra el pseudocódigo de un modelo de aplicación de servidor MQI:

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

```
.
Close queue/Disconnect
.
END
```

El programa de ejemplo CSQ4ICB3 muestra la implementación, en C/370, de un BMP utilizando este modelo. El programa establece la comunicación con IMS primero y, después, con IBM MQ:

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

Return
```

La inicialización de IMS determina si se ha llamado el programa como un BMP orientado a mensajes u orientado a lotes y controla la conexión del gestor de colas IBM MQ y los descriptores de cola en consecuencia.

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

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

Return to calling function
```

La inicialización de IBM MQ se conecta al gestor de colas y abre las colas. En un BMP orientado a mensajes, este se invoca después de que se tome cada punto de sincronización de IMS: en un BMP orientado a lotes, este se invoca solo durante el inicio del programa:

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

```
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

La implementación del modelo de servidor en un MPP (programa de procesamiento de mensajes) se ve influenciada por el hecho de que MPP procesa una única unidad de trabajo por invocación. Esto se debe a que, cuando se toma un punto de sincronización (GU), se cierran la conexión y los manejadores de cola, y se entrega el siguiente mensaje IMS. Esta limitación puede superarse en parte de una de las maneras siguientes:

- **Procesando muchos mensajes en una única unidad de trabajo.**

  Esto implica:

  – Leer un mensaje.
  – Procesar las actualizaciones necesarias.
  – Colocar la respuesta.

  en un bucle hasta que todos los mensajes se hayan procesado o hasta que se haya procesado el número máximo de mensajes, momento en el que se tomará un punto de sincronización.

  Este enfoque solo es válido para determinados tipos de aplicación (por ejemplo, una simple actualización o consulta de base de datos). Aunque los mensajes de respuesta MQI se pueden colocar con la autorización del originador del mensaje MQI que se está manejando, las implicaciones de seguridad de cualquier actualización de recurso IMS se deben abordar con cuidado.

- **Procesando un mensaje por invocación del MPP y garantizando una planificación múltiple del MPP para que procese todos los mensajes disponibles.**

  Utilice el programa de supervisor desencadenante de IBM MQ IMS (CSQQTRMN) para planificar la transacción MPP cuando haya mensajes en la cola de IBM MQ y no haya aplicaciones que le estén prestando servicio.

  Si el supervisor desencadenante inicia el MPP, el nombre del gestor de colas y el nombre de cola se pasan al programa, tal como se muestra en el siguiente fragmento de COBOL:

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

El modelo de servidor, que se espera que sea una tarea de larga ejecución, se soporta mejor en una región de procesamiento por lotes, aunque la BMP no se pueda desencadenar utilizando CSQQTRMN.

Aplicaciones de consulta

Una aplicación típica de IBM MQ que inicia una consulta o actualización funciona de la forma siguiente:

- Recopila datos del usuario.
- Coloca uno o más mensajes de IBM MQ.
- Obtiene los mensajes de respuesta (es posible que tenga que esperarlos)
- Proporciona una respuesta al usuario.

Puesto que los mensajes colocados en colas IBM MQ no pasan a estar disponibles en otras aplicaciones IBM MQ mientras no se confirman, se deben sacar del punto de sincronización, o la aplicación IMS se debe dividir en dos transacciones.

Si la consulta implica colocar un único mensaje, puede utilizar la opción *sin punto de sincronización*; sin embargo, si la consulta es más compleja, o si hay actualizaciones de recursos implicadas, puede que tenga problemas de coherencia si se produce un error y no utiliza puntos de sincronización.

Para superarlo, puede dividir las transacciones MPP de IMS utilizando llamadas MQI utilizando un conmutador de mensajes de programa a programa; consulte *IMS Intersystem Communication (ISC)* para obtener información sobre esto. Esto permite que un programa de consulta se implemente en un MPP:

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

z/OS *Escritura de aplicaciones puente IMS*

Este tema contiene información sobre cómo escribir aplicaciones para utilizar el puente IBM MQ - IMS.

Para obtener información sobre el puente IBM MQ - IMS, consulte El puente IMS.

Utilice los siguientes enlaces para obtener más información sobre la escritura de aplicaciones puente IMS en IBM MQ for z/OS:

- “Cómo el puente IMS maneja los mensajes” en la página 77
- “Escribir programas de transacciones IMS mediante IBM MQ” en la página 934

Conceptos relacionados

“Escritura de aplicaciones IMS utilizando IBM MQ” en la página 72
Hay consideraciones adicionales cuando se utiliza IBM MQ en aplicaciones IMS. Estas incluyen las llamadas de API de MQ que se pueden utilizar y el mecanismo utilizado para el punto de sincronización.

z/OS *Cómo el puente IMS maneja los mensajes*

Cuando utiliza el puente IBM MQ - IMS para enviar mensajes a una aplicación IMS, debe crear sus mensajes con un formato especial.

También debe colocar los mensajes en las colas IBM MQ que se han definido con una clase de almacenamiento que especifica el grupo XCF y el nombre de miembro del sistema IMS de destino. Esto se conoce como colas puente MQ-IMS o simplemente como colas **puente**.

El puente IBM MQ-IMS requiere acceso de entrada exclusivo (MQOO_INPUT_EXCLUSIVE) a la cola puente, si se ha definido con QSGDISP(QMGR), o si se ha definido con QSGDISP(SHARED) junto con la opción NOSHARE.

Un usuario no necesita iniciar sesión en IMS antes de enviar mensajes a una aplicación IMS. El ID de usuario del campo *UserIdentifier* de la estructura MQMD se utiliza para comprobar la seguridad. El nivel de comprobación se determina cuando IBM MQ se conecta a IMS y puede encontrar su descripción en la sección Control de acceso de aplicación para el puente IMS. Esto permite implementar un seudoinicio de sesión.

El puente IBM MQ - IMS acepta los siguientes tipos de mensajes:

- Mensajes que contienen datos de transacción de IMS y una estructura MQIIH. (Se describe en la sección MQIIH):

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

  Nota:

  1. Los corchetes, [ ], representan varios segmentos opcionales.
  2. Establezca el campo *Format* de la estructura MQMD en MQFMT_IMS para utilizar la estructura MQIIH.

- Mensajes que contienen datos de transacciones IMS pero no la estructura MQIIH:

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

IBM MQ valida los datos del mensaje para asegurarse de que la suma de LL bytes más la longitud de MQIIH, si está presente, sea igual a la longitud del mensaje.

Cuando el puente IBM MQ - IMS obtiene mensajes de las colas puente, los procesa de este modo:

- Si el mensaje contiene una estructura MQIIH, el puente verifica el MQIIH, (consulte MQIIH), crea las cabeceras OTMA y envía el mensaje a IMS. El código de transacción se especifica en el mensaje de entrada. Si este es un LTERM, IMS responde con un mensaje DFS1288E. Si el código de transacción representa un mandato, IMS ejecuta el mandato. De lo contrario, el mensaje se pone en cola en IMS para la transacción.
- Si el mensaje contiene datos de la transacción IMS pero no así la estructura MQIIH, el puente IMS presupone lo siguiente:
  - El código de transacción está en los bytes 5 a 12 de los datos de usuario
  - La transacción está en modo no de conversación
  - La transacción está en el modo de confirmación 0 (confirmar y luego enviar)
  - Se utiliza el valor de *Format* de MQMD como *MFSMapName* (en la entrada)
  - El modo de seguridad es MQISS_CHECK

  El mensaje de respuesta también se crea sin una estructura MQIIH, tomando el valor de *Format* para MQMD del *MFSMapName* de la salida de IMS.

El puente IBM MQ - IMS utiliza uno o dos Tpipes para cada cola IBM MQ:

- Se utiliza un Tpipe sincronizado para todos los mensajes que utilizan la modalidad de confirmación 0 (COMMIT_THEN_SEND) (estos se muestran con SYN en el campo de estado del mandato IMS /DIS TMEMBER client TPIPE xxxx)
- Se utiliza un Tpipe no sincronizado para todos los mensajes que utilizan el modo de confirmación 1 (SEND_THEN_COMMIT)

IBM MQ crea los Tpipes cuando se utilizan por primera vez. Un Tpipe existe hasta que se reinicia IMS. Los Tpipes sincronizados existen hasta que IMS se inicia en frío. No puede suprimir estos Tpipes manualmente.

Consulte los temas siguientes para obtener más información acerca de cómo maneja los mensajes el puente IBM MQ - IMS:

- “Correlación de mensajes de IBM MQ con los tipos de transacción IMS” en la página 78
- “Si el mensaje no se puede poner en la cola de IMS” en la página 79
- “Códigos feedback del puente IMS” en la página 79
- “Los campos MQMD de los mensajes del puente IMS” en la página 80
- “Los campos MQIIH de los mensajes del puente IMS” en la página 81
- “Mensajes de respuesta de IMS” en la página 82
- “Utilización de PCB de respuestas alternativas en las transacciones IMS” en la página 82
- “Envío de mensajes no solicitados desde IMS” en la página 82
- “Segmentación de mensajes” en la página 83
- “Conversión de datos para mensajes a y desde el puente IMS” en la página 83

Conceptos relacionados

“Escribir programas de transacciones IMS mediante IBM MQ” en la página 934
El código necesario para manejar las transacciones IMS mediante IBM MQ depende del formato de mensaje que requiera la transacción IMS y el rango de respuestas que pueda devolver. No obstante, hay varios puntos a tener en cuenta cuando su aplicación maneja información con formato de pantalla IMS.

z/OS *Correlación de mensajes de IBM MQ con los tipos de transacción IMS*

Una tabla que describe la correlación de mensajes de IBM MQ con los tipos de transacción IMS.

Tabla 132. Cómo se correlacionan los mensajes de IBM MQ con los tipos de transacción de IMS

Tipo de mensaje de IBM MQ	**Confirmar-luego-enviar (modalidad 0): utiliza Tpipes de IMS sincronizados**	**Enviar-luego-confirmar (modalidad 1): utiliza Tpipes de IMS no sincronizados**
Mensajes de IBM MQ persistentes	• Transacciones de funciones completas recuperables • Las transacciones no recuperables son rechazadas por IMS	• Transacciones de vía de acceso rápida • Transacciones conversacionales • Transacciones de funciones completas
Mensajes de IBM MQ no persistentes	• Transacciones de funciones completas no recuperables • Las transacciones recuperables están permitidas con IMS V8 y APAR PQ61404 y todas las versiones posteriores de IMS	• Transacciones de vía de acceso rápida • Transacciones conversacionales • Transacciones de funciones completas

Nota: Los mandatos IMS no pueden utilizar mensajes IBM MQ persistentes con la modalidad de confirmación 0. Consulte Modalidad de confirmación (commitMode) para obtener más información.

z/OS *Si el mensaje no se puede poner en la cola de IMS*

Obtenga información sobre las acciones que se deben llevar a cabo si el mensaje no se puede poner en la cola de IMS.

Si el mensaje no se puede poner en la cola de IMS, IBM MQ toma la siguiente acción:

- Si un mensaje no se puede poner en IMS porque el mensaje no es válido, el mensaje se coloca en la cola de mensajes no entregados, y se envía un mensaje a la consola del sistema.
- Si el mensaje es válido, pero es rechazado por IMS, IBM MQ envía un mensaje de error a la consola del sistema, el mensaje incluye el código de detección de IMS y el mensaje de IBM MQ se coloca en la cola de mensajes no entregados. Si el código de detección de IMS es 001A, IMS envía un mensaje de IBM MQ que contiene el motivo de la anomalía en la cola de respuestas.

  Nota: En las circunstancias indicadas anteriormente, si IBM MQ no puede colocar el mensaje en la cola de mensajes no entregados por cualquier motivo, el mensaje se devuelve a la cola de IBM MQ de origen. Se envía un mensaje de error a la consola del sistema, y no se envían más mensajes desde esa cola.

  Para reenviar los mensajes, realice ***una*** de las acciones siguientes:

  – Detenga y reinicie Tpipes en el IMS correspondiente a la cola
  – Altere la cola a GET(DISABLED), y otra vez a GET(ENABLED)
  – Detenga y reinicie IMS o el OTMA
  – Detenga y reinicie el subsistema de IBM MQ
- Si IMS rechaza el mensaje para cualquier otra cosa que no sea un error de mensaje, el mensaje de IBM MQ se devuelve a la cola de origen, IBM MQ deja de procesar la cola y se envía un mensaje de error a la consola del sistema.

  Si hace falta un mensaje de informe de excepción, el puente lo coloca en la cola de respuesta con la autoridad del originador. Si no se puede poner el mensaje en la cola, el mensaje de informe se coloca en la cola de mensajes no entregados con la autorización del puente. Si no se puede poner en el DLQ, se descarta.

z/OS *Códigos feedback del puente IMS*

Normalmente, la salida de los códigos de detección IMS es en formato hexadecimal en los mensajes de la consola de IBM MQ, tal como CSQ2001I. Por ejemplo, el código de detección 0x001F. Los códigos feedback de IBM MQ como se muestran en la cabecera de los mensajes no entregados transferidos a la cola de mensajes no entregados son números decimales.

Los códigos de detección del puente IMS están en el rango de 301 a 399, o de 600 a 855 para el código de detección NACK 0x001A. Se correlacionan desde el código de detección IMS-OTMA de este modo:

1. El código de detección IMS-OTMA se convierte de un número hexadecimal a un número decimal.
2. 300 se añade al número resultante del cálculo de 1, lo que proporciona el código de IBM MQ *Feedback*.
3. El código de detección 0x001A de IMS-OTMA, decimal 26 es un caso especial. Se genera un código *Feedback* en el rango de 600-855.
   a. El código de razón IMS-OTMA se convierte de un número hexadecimal a un número decimal.
   b. 600 se añade al número resultante del cálculo de a, lo que proporciona el código de IBM MQ *Feedback*.

Para obtener información sobre los códigos de detección de IMS-OTMA, consulte Códigos de detección de OTMA para mensajes NAK.

z/OS *Los campos MQMD de los mensajes del puente IMS*

Obtenga información sobre los campos MQMD en los mensajes del puente IMS.

El MQMD del mensaje de origen es transportado por IMS en la sección Datos de usuario de las cabeceras de OTMA. Si el mensaje se origina en IMS, el MQMD se construye mediante la salida de resolución de destino de IMS. El MQMD de un mensaje recibido de IMS se crea de la forma siguiente:

StrucID
"MD "

Versión
MQMD_VERSION_1

Informe
MQRO_NONE

MsgType
MQMT_REPLY

Caducidad
Si MQIIH_PASS_EXPIRATION se establece en el campo Distintivos de MQIIH, este campo contiene el tiempo de caducidad restante; de lo contrario, se establece en MQEI_UNLIMITED

Comentarios
MQFB_NONE

Encoding
MQENC.Native (la codificación del sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (el ID del juego de caracteres codificados del sistema z/OS)

Formato
MQFMT_IMS si el MQMD.Format del mensaje de entrada es MQFMT_IMS, de lo contrario, IOPCB.MODNAME

Prioridad
MQMD.Priority del mensaje de entrada

Persistence
Depende de la modalidad de confirmación: MQMD.Persistence del mensaje de entrada si CM-1; la persistencia es igual a la recuperabilidad del mensaje de IMS si CM-0

MsgId
MQMD.MsgId si MQRO_PASS_MSG_ID, de lo contrario, nuevo MsgId (el valor predeterminado)

CorrelId
MQMD.CorrelId del mensaje de entrada si MQRO_PASS_CORREL_ID, de lo contrario, MQMD.MsgId del mensaje de entrada (valor predeterminado)

BackoutCount
0

ReplyToQ
Espacios en blanco

ReplyToQMgr
Espacios en blanco (establecido en el nombre qmgr local por el gestor de colas durante la MQPUT)

UserIdentifier
MQMD.UserIdentifier del mensaje de entrada

AccountingToken
MQMD.AccountingToken del mensaje de entrada

ApplIdentityData
MQMD.ApplIdentityData del mensaje de entrada

PutApplType
MQAT_XCF si no hay ningún error, de lo contrario MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> si no hay ningún error, de lo contrario, el nombre QMGR

PutDate
Fecha cuando se colocó el mensaje

PutTime
Hora cuando se colocó el mensaje

ApplOriginData
Espacios en blanco

z/OS *Los campos MQIIH de los mensajes del puente IMS*

Obtenga información sobre los campos MQIIH en los mensajes del puente IMS.

El MQIIH de un mensaje recibido de IMS se crea de la forma siguiente:

StrucId
"IIH "

Versión
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat del mensaje de entrada si MQIIH.ReplyToFormat no está en blanco; de lo contrario, IOPCB.MODNAME

Flags
0

LTermOverride
Nombre de LTERM (Tpipe) de la cabecera OTMA

MFSMapName
Nombre de correlación de la cabecera OTMA

ReplyToFormat
Espacios en blanco

Authenticator
MQIIH.Authenticator del mensaje de entrada si el mensaje de respuesta se está transfiriendo a una cola puente de MQ-IMS, de lo contrario, se deja en blanco.

TranInstanceId
ID de conversación / Señal de servidor de la cabecera OTMA, si está en la conversación. En las versiones de IMS anteriores a V14, este campo siempre adopta un valor nulo si no está en la conversación. A partir de IMS V14 hacia adelante, este campo puede ser establecido por IMS aunque no esté en la conversación.

TranState
"C" si está en conversación, de lo contrario, en blanco

CommitMode
Modalidad de compromiso de la cabecera OTMA ("0" o "1")

SecurityScope
Espacio en blanco

Reserved
Espacio en blanco

z/OS *Mensajes de respuesta de IMS*

Cuando una transacción IMS realiza una llamada ISRT a su IOPCB, el mensaje se direcciona de nuevo al LTERM o TPIPE de origen.

Estos se ven en IBM MQ como mensajes de respuesta. Los mensajes de respuesta de IMS se colocan en la cola de respuestas especificada en el mensaje original. Si el mensaje no puede colocarse en la cola de respuesta, se coloca en la cola de mensajes no entregados utilizando la autorización del puente. Si el mensaje no puede colocarse en la cola de mensajes no entregados, se envía un acuse de recibo negativo a IMS indicando que no se puede recibir el mensaje. La responsabilidad del mensaje se devuelve entonces a IMS. Si utiliza la modalidad de confirmación 0, los mensajes desde ese Tpipe no se envían al

puente y permanecen en la cola de IMS, es decir, no se envían más mensajes hasta que se reinicie. Si utiliza la modalidad de confirmación 1, otros trabajos pueden continuar.

Si la respuesta tiene una estructura MQIIH, su tipo de formato es MQFMT_IMS; de lo contrario, su tipo de formato se especifica mediante el nombre de MOD de IMS que se utiliza al insertar el mensaje.

z/OS *Utilización de PCB de respuestas alternativas en las transacciones IMS*

Cuando una transacción IMS utiliza PCB de respuestas alternativas (realiza una llamada ISRT al ALTPCB o emite una llamada CHNG a un PCB modificable), se invoca la salida previa al direccionamiento (DFSYPRX0) para determinar si el mensaje debe redireccionarse.

Si el mensaje se va a redireccionar, la salida de resolución de destino (DFSYDRU0) se invoca para confirmar el destino y preparar la información de cabecera. Consulte Utilización de salidas OTMA en IMS y La salida previa al direccionamiento DFSYPRX0 para obtener más información sobre estos programas de salida.

A menos que se realice una acción en las salidas, toda la salida de las transacciones IMS iniciadas desde un gestor de colas de IBM MQ, tanto a IOPCB como a ALTPCB, se devolverán al mismo gestor de colas.

z/OS *Envío de mensajes no solicitados desde IMS*

Para enviar mensajes desde IMS a una cola IBM MQ, tiene que invocar una transacción IMS que realice ISRT en un ALTPCB.

Tiene que escribir salidas de resolución de destino predireccionamiento para direccionar mensajes no solicitados desde IMS y crear los datos de usuario OTMA, para que el MQMD del mensaje se pueda crear correctamente. Consulte Salida de predireccionamiento DFSYPRX0 y Salida de usuario de resolución de destino para obtener información sobre estos programas de salida.

Nota: El puente IBM MQ - IMS no sabe si un mensaje que recibe es una respuesta o un mensaje no solicitado. Maneja el mensaje de la misma forma en cada caso, creando el MQMD y la MQIIH de la respuesta a partir del UserData de OTMA que llegan en el mensaje

Los mensajes no solicitados pueden crear nuevos Tpipes. Por ejemplo, si una transacción IMS existente ha conmutado a un nuevo LTERM (por ejemplo PRINT01), pero la implementación requiere que la salida se entregue a través de OTMA, se crea un nuevo Tpipe (llamado PRINT01 en este ejemplo). De forma predeterminada, se trata de un Tpipe no sincronizado. Si la implementación requiere que el mensaje sea recuperable, establezca el distintivo de salida de la salida de resolución de destino. Consulte la *Guía de personalización de IMS* para obtener más información.

z/OS *Segmentación de mensajes*

Puede definir las transacciones de IMS de modo que esperen una entrada de un único segmento o de varios segmentos.

La aplicación de IBM MQ de origen debe crear la entrada se salida utilizando la siguiente estructura MQIIH como uno o varios segmentos de datos LLZZ. Todos los segmentos de un mensaje de IMS deben estar contenidos en un único mensaje de IBM MQ que se envía con una única llamada MQPUT.

La longitud máxima de un segmento de datos LLZZ se define mediante IMS/OTMA (32767 bytes). La longitud total del mensaje de IBM MQ es la suma de todos los bytes LL más la longitud de la estructura MQIIH.

Todos los segmentos de la respuesta están contenidos en un único mensaje de IBM MQ.

Existe una restricción adicional en la limitación de 32 KB de los mensajes con el formato MQFMT_IMS_VAR_STRING. Cuando los datos de mensaje CCSID con ASCII combinado se convierten en un mensaje CCSID con EBCDIC combinado, se añade un byte de desplazamiento a teclado o un byte de desplazamiento desde teclado cada vez que hay una transacción entre los caracteres SBCS y DBCS. La restricción de 32 KB se aplica al tamaño máximo del mensaje. Esto es, dado que el campo LL del mensaje no puede superar los 32 KB, el mensaje no puede superar los 32 KB incluidos los caracteres de desplazamiento a o desde teclado. La aplicación que crea el mensaje debe tener esto en cuenta.

z/OS *Conversión de datos para mensajes a y desde el puente IMS*

La conversión de datos se lleva a cabo mediante la función de gestión de colas distribuidas, que puede llamar a cualquier salida necesaria, o mediante el agente de transferencia a colas dentro del grupo, que no da soporte al uso de salidas, cuando transfiere un mensaje a una cola de destino que tiene definida información XCF para su clase de almacenamiento. La conversión de datos no se produce cuando se entrega un mensaje a una cola mediante publicación/suscripción.

Cualquier salida debe estar disponible para el recurso de gestión de colas distribuidas del conjunto de datos al que hace referencia la sentencia CSQXLIB DD. Esto significa que puede enviar mensajes a una aplicación IMS utilizando el puente IBM MQ - IMS desde cualquier plataforma de IBM MQ.

Si hay errores de conversión, el mensaje se transfiere a la cola sin conversión. Finalmente, esto hace que el puente IBM MQ - IMS lo considere un error, ya que el puente no puede reconocer el formato de cabecera. Si se produce un error de conversión, el mensaje se envía a la consola de z/OS.

Consulte la sección “Escribir salidas de conversión de datos” en la página 1009 para obtener información detallada acerca de la conversión de datos en general.

Envío de mensajes al puente IBM MQ - IMS

Para asegurarse de que la conversión se realiza correctamente, debe indicar al gestor de colas el formato del mensaje.

Si el mensaje tiene una estructura MQIIH, el campo *Format* de MQMD debe establecerse en el formato incluido MQFMT_IMS y el campo *Format* de MQIIH debe establecerse en el nombre del formato que describe los datos de su mensaje. Si no existe ningún MQIIH, establezca el campo *Format* de MQMD en su nombre de formato.

Si sus datos, excepto LLZZ, son todos datos de caracteres (MQCHAR), utilice como nombre de formato, en MQIIH o MQMD, según convenga, el formato MQFMT_IMS_VAR_STRING incluido. De lo contrario, utilice su propio nombre de formato, en cuyo caso debe proporcionar también la salida de conversión de datos para su formato. La salida debe manejar la conversión de los LLZZ de su mensaje, además de los propios datos, pero no tiene que manejar ningún MQIIH en el inicio del mensaje.

Si su aplicación utiliza *MFSMapName*, en su lugar, puede utilizar mensajes con MQFMT_IMS y definir el nombre de correlación que se ha pasado a la transacción IMS en el campo MFSMapName de MQIIH.

Recepción de mensajes desde el puente IBM MQ - IMS

Si está presente una estructura MQIIH en el mensaje original que envía a IMS, también estará presente una estructura en el mensaje de respuesta.

Para asegurarse de que la respuesta se convierte correctamente:

- Si tiene una estructura MQIIH en su mensaje original, especifique el formato que desea para su mensaje de respuesta en el campo *ReplytoFormat* de MQIIH del mensaje original. Este valor se sustituye en el campo *Format* de MQIIH del mensaje de respuesta. Esto resulta especialmente útil si todos sus datos tienen el formato LLZZ<datos caracteres>.
- Si no tiene una estructura MQIIH en su mensaje original, especifique el formato que desea para el mensaje de respuesta, como el nombre MFS MOD del ISRT de la aplicación IMS en IOPCB.

z/OS *Escribir programas de transacciones IMS mediante IBM MQ*

El código necesario para manejar las transacciones IMS mediante IBM MQ depende del formato de mensaje que requiera la transacción IMS y el rango de respuestas que pueda devolver. No obstante, hay varios puntos a tener en cuenta cuando su aplicación maneja información con formato de pantalla IMS.

Cuando se inicia una transacción IMS desde una pantalla 3270, el mensaje pasa a través de los servicios de formato de mensajes de IMS. Esto puede eliminar toda la dependencia del terminal que tiene la corriente de datos que ve la transacción. Cuando se inicia una transacción mediante OTMA, MFS no está implicado. Si se implementa la lógica de la aplicación en MFS, se debe volver a crear en la nueva aplicación.

En algunas transacciones de IMS, la aplicación de usuario final puede determinar algún comportamiento de pantalla 3270, por ejemplo, el resaltado de un campo en el que se han especificado datos no válidos. Este tipo de información se comunica añadiendo un campo de atributo de dos bytes en el mensaje de IMS para cada campo de pantalla que debe modificar el programa.

De este modo, si está codificando una aplicación para que sea similar a 3270, debe tener en cuenta estos campos al crear o recibir mensajes.

Es posible que necesite codificar la información de su programa para procesar:

- La tecla que se ha de pulsar, por ejemplo, Intro y PF1
- La ubicación del cursor cuando se pasa el mensaje a su aplicación
- Si los campos de atributos los ha establecido la aplicación IMS
  - La intensidad alta, normal o cero
  - El color
  - Si IMS espera la devolución del campo la próxima que se pulse Intro
- Si la aplicación IMS ha utilizado caracteres nulos (X'3F') en cualquier campo.

Si su mensaje IMS solo contiene datos de caracteres, además del segmento LLZZ-data, y está utilizando una estructura MQIIH, establezca el formato MQMD en MQFMT_IMS y el formato MQIIH en MQFMT_IMS_VAR_STRING.

Si su mensaje IMS solo contiene datos de caracteres, además del segmento LLZZ-data, y **no** está utilizando una estructura MQIIH, establezca el formato MQMD en MQFMT_IMS_VAR_STRING y asegúrese de que su aplicación IMS especifica MODname MQFMT_IMS_VAR_STRING en la respuesta. Si se produce un problema (por ejemplo, un usuario no autorizado a utilizar la transacción) y IMS envía un mensaje de error, este tiene un MODname con el formato DFSMOx, donde x es un número comprendido entre 1 y 5. Esto se coloca en el MQMD de MQMD.Format.

Si su mensaje IMS contiene datos binarios, empaquetados o de punto flotante, además del segmento LLZZ-data, codifique sus propias rutinas de conversión de datos. Consulte la publicación *IMS/ESA Application Programming: Transaction Manager* para obtener más información acerca de los formatos de pantalla de IMS.

Tenga en cuenta los temas siguientes cuando codifique cómo manejar las transacciones IMS mediante IBM MQ.

- "Escribir aplicaciones IBM MQ para invocar transacciones de conversación IMS" en la página 935
- "Escribir programas que contienen mandatos IMS" en la página 936
- "Desencadenamiento" en la página 936

Escribir aplicaciones IBM MQ para invocar transacciones de conversación IMS

Utilice esta información como guía de los puntos a tener en cuenta cuando escriba una aplicación IBM MQ para invocar transacciones de conversación de IMS.

Cuando escribe una aplicación que invoca una conversación de IMS, tenga en cuenta lo siguiente:

- Incluya una estructura MQIIH en su mensaje de aplicación.
- Establezca *CommitMode* de MQIIH en MQICM_SEND_THEN_COMMIT.
- Para invocar una nueva conversación, establezca el campo *TranState* de MQIIH en MQITS_NOT_IN_CONVERSATION.
- Para invocar el segundo paso y los siguientes de una conversación, establezca *TranState* en MQITS_IN_CONVERSATION y establezca *TranInstanceId* en el valor del campo que se ha devuelto en el paso anterior de la conversación.
- No hay un modo fácil en IMS de encontrar el valor de *TranInstanceId*, si pierde el mensaje original enviado desde IMS.

- La aplicación debe comprobar el *TranState* de los mensajes de IMS para comprobar si la transacción IMS ha finalizado la conversación.
- Puede utilizar /EXIT para finalizar una conversación. También debe entrecomillar *TranInstanceId*, establecer *TranState* en MQITS_IN_CONVERSATION y utilizar la cola IBM MQ en la que se está llevando a cabo la conversación.
- No puede utilizar /HOLD o /REL para retener o liberar una conversación.
- Las conversaciones que se invocan mediante el puente IBM MQ - IMS finalizan si se reinicia IMS.

Escribir programas que contienen mandatos IMS

Un programa de aplicación puede crear un mensaje de IBM MQ con el formato LLZZ*mandato*, en lugar de una transacción, donde *mandato* tiene el formato /DIS TRAN PART o /DIS POOL ALL.

La mayor parte de los mandatos IMS se pueden emitir de este modo. Consulte *IMS V11 Communications and Connections* para obtener más detalles. La salida del mandato se recibe en el mensaje de respuesta de IBM MQ con formato de texto, tal como se envía a un terminal 3270 para visualizarla.

OTMA implementa un formato especial del mandato de transacción de visualización de IMS que devuelve un formato de salida con una arquitectura. El formato exacto está definido en el documento *IMS V11 Communications and Connections*. Para invocar este formato desde un mensaje de IBM MQ, previamente cree los datos del mensaje como antes, por ejemplo, /DIS TRAN PART, y establezca el campo TranState de MQIIH en MQITS_ARCHITECTED. IMS procesa el mandato y devuelve la respuesta con el formato de arquitectura. Una respuesta de arquitectura contiene toda la información que se puede encontrar en el formato de texto de la salida y un fragmento de información adicional: Si la transacción se ha definido como recuperable o no recuperable.

Desencadenamiento

El puente IBM MQ - IMS no da soporte a los mensajes de desencadenantes.

Si define una cola de iniciación que utiliza una clase de almacenamiento con parámetros XCF, se rechazarán los mensajes que se coloquen en dicha cola cuando lleguen al puente.

Desarrollo de aplicaciones procedimentales cliente

Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

Las aplicaciones se pueden crear y ejecutar en el entorno de cliente de IBM MQ. La aplicación se tiene que compilar y enlazar con el IBM MQ MQI client usado. La forma en que se compilan y enlazan las aplicaciones varía en función de la plataforma y del lenguaje de programación utilizado. Para obtener información sobre cómo compilar aplicaciones cliente, consulte “Creación de aplicaciones para IBM MQ MQI clients” en la página 942.

Puede ejecutar una aplicación de IBM MQ en un entorno de IBM MQ completo y en un entorno de IBM MQ MQI client sin cambiar el código, siempre que se cumplan determinadas condiciones. Para obtener más información sobre cómo ejecutar aplicaciones en el entorno de cliente de IBM MQ, consulte “Ejecución de aplicaciones en el entorno de IBM MQ MQI client” en la página 944.

Si se usa la interfaz de cola de mensajes (MQI) para desarrollar aplicaciones que ejecuten en un entorno IBM MQ MQI client, hay que imponer algunos controles adicionales durante una llamada MQI para garantizar que el procesamiento de la aplicación IBM MQ no se vea afectado. Para obtener más información sobre estos controles, consulte “Utilización de MQI en una aplicación cliente” en la página 937.

Consulte los temas siguientes para obtener información sobre la preparación y ejecución de otros tipos de aplicaciones como aplicaciones cliente:

- “Preparación y ejecución de las aplicaciones CICS y Tuxedo” en la página 957
- “Preparación y ejecución de aplicaciones Microsoft Transaction Server” en la página 50
- “Preparación y ejecución de aplicaciones IBM MQ JMS” en la página 960

Conceptos relacionados

"Conceptos de desarrollo de aplicaciones" en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

"Desarrollo de aplicaciones para IBM MQ" en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

"Consideraciones acerca del diseño de las aplicaciones IBM MQ" en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

"Desarrollo de una aplicación procedimental para encolamientos" en la página 740
Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

"Escritura de aplicaciones de publicación/suscripción" en la página 829
Empezar a escribir aplicaciones de publicación/suscripción de IBM MQ.

"Creación de una aplicación procedimental" en la página 1025
Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

"Tratamiento de errores en un programa procedimental" en la página 1064
Esta información explica los errores asociados a las llamadas MQI de una aplicación cuando se efectúa una llamada o cuando el mensaje se entrega en su destino final.

Tareas relacionadas

"Utilización de programas procedimentales de ejemplo de IBM MQ" en la página 1085
Estos programas de ejemplo se escriben en lenguajes de procedimiento y muestran los usos típicos de la interfaz de cola de mensajes (MQI). Programas IBM MQ en distintas plataformas.

Utilización de MQI en una aplicación cliente

Esta colección de temas considera las diferencias entre la escritura de la aplicación de IBM MQ para ejecutarse en un entorno de cliente de interfaz de cola de mensajes (MQI) y para ejecutarse en el entorno de gestor de colas de IBM MQ completo.

Cuando diseñe una aplicación, tenga en cuenta los controles que se deben imponer durante una llamada MQI para asegurarse de que el proceso de aplicaciones de IBM MQ no se interrumpa.

Para poder ejecutar aplicaciones que utilicen la MQI, debe crear determinados objetos de IBM MQ. Para obtener más información, consulte Programas de aplicación que utilizan la MQI.

Limitación del tamaño de un mensaje en una aplicación cliente

Un gestor de colas tiene una longitud de mensaje máxima, pero el tamaño máximo del mensaje que puede transmitir desde una aplicación cliente está limitado por la definición de canal.

El atributo de longitud máxima de mensaje (MaxMsgLength) de un gestor de colas indica la longitud máxima de un mensaje que puede manejar ese gestor de colas.

Multi En Multiplatforms, puede aumentar el atributo de longitud máxima de mensaje de un gestor de colas. Para obtener más información, consulte ALTER QMGR.

Puede determinar el valor del atributo MaxMsgLength para un gestor de colas utilizando la llamada MQINQ.

Si se cambia el atributo MaxMsgLength, no se comprueba si existen colas, e incluso mensajes, con una longitud mayor que el valor nuevo. Después de cambiar este atributo, reinicie las aplicaciones y los canales para asegurarse de que el cambio ha entrado en vigor. De esta forma no será posible que se creen

mensajes nuevos con una longitud que sea mayor que el valor MaxMsgLength del gestor de colas o de la cola (a menos que se permita la segmentación del gestor de colas).

La longitud máxima de mensaje contenida en una definición de canal limita el tamaño de los mensajes que se pueden transmitir a través de una conexión de cliente. Si una aplicación de IBM MQ intenta utilizar la llamada MQPUT o la llamada MQGET con un mensaje mayor que ese valor, se devuelve un código de error a la aplicación. El parámetro de tamaño máximo de mensaje contenido en la definición de canal no afecta al tamaño máximo de mensaje que se puede consumir utilizando MQCB a través de una conexión de cliente.

Conceptos relacionados

"Utilización de MQCONNX" en la página 942
Puede emplear la llamada MQCONNX para especificar una estructura de definición de canal (MQCD) en la estructura MQCNO.

Referencia relacionada

Longitud máxima de mensaje (MAXMSGL)

ALTER CHANNEL

2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Seleccionar el CCSID de cliente o servidor

Utilice el identificador de conjunto de caracteres codificados (CCSID) para el cliente. El gestor de colas realiza la conversión necesaria. Puede utilizar la variable de entorno **MQCCSID** para alterar temporalmente el CCSID. Si la aplicación realiza varias operaciones PUT, el CCSID y los campos de codificación de MQMD se pueden sobrescribir cuando se ha finalizado la primera operación PUT.

Los datos que se pasan por la interfaz de cola de mensajes (MQI) desde la aplicación al apéndice de cliente deben estar en el CCSID local y codificados para el IBM MQ MQI client. Si el gestor de colas conectado requiere la conversión de datos, la conversión se hará mediante el código de soporte del cliente en el gestor de colas.

En IBM WebSphere MQ 7.0 y versiones posteriores, el cliente Java puede realizar la conversión si el gestor de colas no puede llevarlo a cabo. Consulte "Conexiones de cliente de IBM MQ classes for Java" en la página 383

El código de cliente presupone que los datos de caracteres que pasan por la MQI en el cliente están en el CCSID configurado para dicha estación de trabajo. Si este CCSID es un CCSID no soportado o no es el CCSID necesario, se puede alterar temporalmente con la variable de entorno **MQCCSID** utilizando uno de estos mandatos:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Si este parámetro se establece en el perfil, se presupone que todos los datos MQI pertenecen a la página de códigos 850.

Nota: La suposición sobre la página de códigos 850 no se aplica a los datos de la aplicación del mensaje.

Si la aplicación está realizando varias operaciones PUT que incluyen cabeceras de IBM MQ después del descriptor de mensaje (MQMD), tenga en cuenta que el CCSID y los campos de codificación de MQMD se sobrescriben cuando finaliza la primera operación PUT.

Después del primer PUT, estos campos contienen el valor que ha utilizado el gestor de colas conectado para convertir las cabeceras de IBM MQ. Asegúrese de que su aplicación restablece los valores a los que se requieran.

Utilización de MQINQ en una aplicación cliente

El código de cliente modifica algunos valores consultados mediante MQINQ.

CCSID
se establece en el CCSID del cliente, no en el del gestor de colas.

MaxMsgLength
se reduce si está limitado por la definición de canal. Será el valor más bajo de los siguientes:

- El valor definido en la definición de cola o
- El valor definido en la definición de canal

Para obtener más información, consulte MQINQ.

Utilización de la coordinación de puntos de sincronización en una aplicación cliente

Una aplicación que se ejecuta en el cliente base puede emitir MQCMIT y MQBACK, pero el ámbito del control de punto de sincronización está limitado a los recursos de MQI. Puede utilizar un gestor de transacciones externo con un cliente transaccional extendido.

En IBM MQ, uno de los roles del gestor de colas es el control de punto de sincronización dentro en una aplicación. Si una aplicación se ejecuta en un cliente base de IBM MQ, puede emitir MQCMIT y MQBACK, pero el ámbito del control de punto de sincronización está limitado a los recursos de MQI. El verbo MQBEGIN de IBM MQ no es válido en un entorno de cliente base.

Las aplicaciones que se ejecutan en el servidor, en un entorno de gestor de colas completo, pueden coordinar varios recursos (por ejemplo, bases de datos) mediante un supervisor de transacciones. En el servidor, puede utilizar el Supervisor de transacciones que se proporciona con los productos IBM MQ u otro supervisor de transacciones como, por ejemplo, CICS. No puede utilizar un supervisor de transacciones con una aplicación de cliente base.

Puede utilizar un gestor de transacciones externo con un cliente transaccional extendido de IBM MQ. Consulte ¿Qué es un cliente transaccional extendido? para obtener información detallada.

Utilización de la lectura anticipada en una aplicación cliente

Puede utilizar la lectura anticipada en un cliente para permitir que se envíen mensajes no persistentes a un cliente sin que la aplicación cliente tenga que solicitarlos.

Cuando un cliente necesita un mensaje de un servidor, envía una petición al servidor. Envía una petición separada para cada uno de los mensajes que consume. Para mejorar el rendimiento de un cliente que consume mensajes no persistentes evitando tener que enviar estos mensajes de petición, un cliente se puede configurar para que utilice lectura anticipada. La lectura anticipada permite enviar mensajes a un cliente sin que una aplicación tenga que solicitarlos.

La utilización de la lectura anticipada puede mejorar el rendimiento al consumir mensajes no persistentes de una aplicación cliente. Esta mejora en el rendimiento está disponible para las aplicaciones MQI y JMS. Las aplicaciones cliente que utilizan MQGET o consumo asíncrono se benefician de las mejoras de rendimiento al consumir mensajes no persistentes.

Cuando se llama a MQOPEN con MQOO_READ_AHEAD, el cliente de IBM MQ sólo permite la lectura anticipada si se cumplen ciertas condiciones. Estas condiciones incluyen:

- La aplicación cliente debe compilarse y enlazarse a las bibliotecas de cliente MQI de IBM MQ con hebras.
- El canal de cliente debe utilizar el protocolo TCP/IP

- El canal debe tener un valor SharingConversations (SHARECNV) distinto de cero tanto en las definiciones de canal de cliente y servidor.

Cuando la lectura anticipada está habilitada, se envían mensajes a un almacenamiento intermedio de memoria interna en el cliente, llamado almacenamiento intermedio de lectura anticipada. El cliente tiene un almacenamiento intermedio de lectura anticipada para cada cola que tenga abierta con lectura anticipada habilitada. Los mensajes del almacenamiento intermedio de lectura anticipada no tienen persistencia. El cliente actualiza periódicamente el servidor con información sobre la cantidad de datos que ha consumido.

No todos los diseños de aplicaciones cliente resultan adecuados para utilizar la lectura anticipada, debido a que no todas las opciones están soportadas para su uso. Algunas opciones son necesarias para ser coherentes entre las llamadas MQGET cuando la lectura anticipada está habilitada. Si un cliente altera sus criterios de selección entre llamadas MQGET, los mensajes que se almacenan en el almacenamiento intermedio de lectura anticipada permanecen abandonados en el almacenamiento intermedio de lectura anticipada del cliente. Para obtener más información, consulte “Mejora del rendimiento de mensajes no persistentes” en la página 807

La configuración de lectura anticipada está controlada por tres atributos, MaximumSize, PurgeTime y UpdatePercentage, que se especifican en la stanza MessageBuffer del archivo de configuración de cliente de IBM MQ.

Utilización de una transferencia asíncrona en una aplicación de cliente

Mediante la transferencia asíncrona, una aplicación puede transferir un mensaje a una cola sin tener que esperar una respuesta del gestor de colas. Puede utilizar esto para mejorar el rendimiento de la mensajería en algunas situaciones.

Normalmente, cuando una aplicación transfiere un mensaje o mensajes a una cola, utilizando MQPUT o MQPUT1, la aplicación tiene que esperar a que el gestor de colas confirme que ha procesado la petición MQI. Puede mejorar el rendimiento de la mensajería, especialmente para aplicaciones que utilizan enlaces de cliente, y aplicaciones que transfieren un gran número de mensajes pequeños a una cola, eligiendo, en su lugar, transferir mensajes de forma asíncrona. Cuando una aplicación transfiere un mensaje asíncronamente, el gestor de colas no devuelve la confirmación de éxito o error de cada llamada, pero el usuario puede comprobar periódicamente la existencia de errores.

Para colocar un mensaje en una cola de forma asíncrona, utilice la opción MQPMO_ASYNC_RESPONSE en el campo *Options* de la estructura MQPMO.

Si un mensaje no cumple las condiciones para la transferencia asíncrona, se transfiere a una cola de forma síncrona.

Cuando se solicita una respuesta de transferencia asíncrona para MQPUT o MQPUT1, un código CompCode y una razón MQCC_OK y MQRC_NONE no significan necesariamente que el mensaje se haya transferido correctamente a una cola. Aunque es posible que no se devuelva inmediatamente un código de error o de éxito de la llamada MQPUT o MQPUT1, el primer error que se produce en una llamada asíncrona se puede determinar posteriormente mediante una llamada MQSTAT.

Para obtener más información sobre MQPMO_ASYNC_RESPONSE, consulte la sección Opciones de MQPMO.

El programa de ejemplo de transferencia asíncrona muestra algunas de las funciones disponibles. Para obtener más detalles acerca de las funciones y el diseño del programa y cómo ejecutarlo, consulte la sección “El programa de ejemplo Asynchronous Put (operación de transferencia asíncrona)” en la página 1105.

Utilización de conversaciones de compartición en una aplicación cliente

En un entorno en el que se permita compartir conversaciones, estas pueden compartir una instancia de canal MQI.

La compartición de conversaciones se controla con dos campos, ambos llamados SharingConversations, uno que forma parte de la estructura de definición de canal (MQCD) y otro que forma parte de la estructura de parámetro de salida de canal (MQCXP). El campo SharingConversations de la estructura

MQCD es un valor entero, que determina el número máximo de conversaciones que pueden compartir una instancia de canal asociada al canal. El campo SharingConversations en MQCXP es un valor booleano que indica si la instancia de canal está compartida en ese momento.

En un entorno en el que la compartición de conversaciones no está permitida, las conexiones de cliente nuevas que especifiquen MQCD idénticas no compartirán una instancia de canal.

Una conexión de aplicación cliente nueva compartirá la instancia de canal cuando se cumplan las siguientes condiciones:

- Los extremos de la instancia de canal de conexión de cliente y de servidor están configurados para compartir conversaciones y estos valores no son sustituidos poor la salidas de canal.
- El valor MQCD de la conexión cliente (suministrado en la llamada MQCONNX cliente o desde la tabla de definiciones de canal de cliente (CCDT) coincide exactamente con el valor MQCD de la conexión cliente suministrado en la llamada MQCONNX cliente o desde la CCDT cuando se establece por primera vez la instancia de canal existente. Recuerde que la MQCD original puede haber sido sustituida por salidas o por negociación del canal, pero que la comparación se realiza con el valor suministrado al sistema cliente antes de realizar estos cambios.
- No se supera el limite de conversaciones de compartición en el lado del servidor.

Si una conexión de aplicación cliente nueva coincide con los criterios de ejecución de compartición de una instancia de canal con otras conversaciones, esta decisión se toma antes de llamar a cualquier salida en esa conversación. Las salidas en dicha conversación no pueden cambiar el hecho de que está compartiendo la instancia de canal con otras conversaciones. Si no existen instancias de canal que coincidan con la definición de canal nueva, se conecta una instancia de canal nueva.

La negociación de canal solo tiene lugar en la primera conversación en una instancia de canal; los valores negociados para la instancia de canal se fijan en ese momento y no podrán modificarse cuando se inicien conversaciones posteriores. La autenticación TLS también se produce únicamente en la primera conversación.

Si se modifica el valor SharingConversations de la MQCD durante la inicialización de cualquier salida de seguridad, emisión o recepción en la primera conversación en el socket en el extremo de las conexiones de cliente o de servidor de la instancia de canal, se utilizará el valor nuevo que tenga una vez inicializadas todas estas salidas para determinar el valor de las conversaciones de compartición de la instancia de canal (el valor más bajo tiene preferencia).

Si el valor negociado para las conversaciones de compartición es cero, la instancia de canal nunca se compartirá. Otros programas de salida que establezcan este campo a cero ejecutarán de forma similar en su propia instancia de canal.

Si el valor negociado para las conversaciones de compartición es mayor de cero, SharingConversations de MQCXP se establecerá a TRUE en las posteriores llamadas a salidas, indicando que se puede entrar en otros programas de salidas en esta instancia de canal de forma simultánea con esta.

Cuando escriba un programa de salida de canal, tenga en cuenta si se va a ejecutar en una instancia de canal que pueda involucrar conversaciones de compartición. Si la instancia de canal puede implicar compartir conversaciones, tenga en cuenta el efecto que la modificación de campos de la MQCD tendrá sobre otras instancias de la salida de canal; todos los campos de la MQCD tienen valores comunes en todas las conversaciones de compartición. Una vez establecida la instancia de canal, si los programas de salida intentan modificar campos de la MQCD pueden tener problemas, porque otras instancias de programas de salida ejecutados en la instancia de canal podrían estar intentando modificar los mismos campos al mismo tiempo. Si esta situación pudiera surgir en sus programas de salida, tendrá que serializar el acceso a la MQCD en el código de la salida.

Si está trabajando con un canal definido para compartir conversaciones, pero no desea que se produzca la compartición en una instancia de canal concreta, establezca el valor de SharingConversations de la MQCD a 1 o 0 cuando inicialice una salida de canal en la primera conversación de la instancia de canal. Consulte SharingConversations para obtener una explicación de los valores de SharingConversations.

Ejemplo

La compartición de conversaciones está habilitada.

Está utilizando una definición de canal de conexión cliente que especifica un programa de salida.

La primera vez que se se inicia este canal, el programa de salida modifica algunos de los parámetros de la MQCD al inicializarse. El canal actúa sobre ellas, así que la definición del canal en el que se está ejecutando es ahora diferente de la suministrada originalmente. El parámetro SharingConversations de la MQCXP está establecido a TRUE.

La próxima vez que la aplicación se conecte utilizando este canal, la conversación se ejecutará en la instancia de canal iniciada anteriormente, porque tiene la misma definición de canal original. La instancia de canal con la que la aplicación se conecta por segunda vez es la misma instancia que cuando se conectó por primera vez. Por consiguiente, utilizará las definiciones modificadas por el programa de salida. Cuando se inicializa el programa de salida para la segunda conversación, aunque pueda modificar los campos de la MQCD, el canal no actúa sobre ellas. Estas mismas características son aplicables a cualquier conversación posterior que comparta la instancia de canal.

Utilización de MQCONNX

Puede emplear la llamada MQCONNX para especificar una estructura de definición de canal (MQCD) en la estructura MQCNO.

Esto permite a la aplicación cliente que realiza la llamada especificar la definición del canal de conexión de cliente en el tiempo de ejecución. Para obtener más información, consulte Creación de un canal de conexión de cliente en IBM MQ MQI client utilizando MQCNO. Cuando se utiliza MQCONNX, la llamada emitida en el servidor depende del nivel del servidor y la configuración del escucha.

Cuando se utiliza MQCONNX desde un cliente, no se tendrán en cuenta las opciones siguientes:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

La estructura MQCD que puede utilizar depende del número de versión de MQCD que emplee. Para obtener información sobre las versiones de MQCD (MQCD_VERSION), consulte Versión de MQCD. Puede utilizar la estructura MQCD, por ejemplo, para pasar programas de salida de canal al servidor. Si utiliza MQCD Versión 3 o posterior, puede utilizar la estructura para pasar una matriz de salidas al servidor. Puede utilizar esta función para realizar más de una operación en el mismo mensaje, por ejemplo, cifrado y compresión, añadiendo una salida para cada operación, en lugar de modificar una salida existente. Si no especifica una matriz en la estructura MQCD, se comprobarán los campos de salida única. Para obtener más información sobre los programas de salida de canal, consulte “Programas de salida de canal para canales de mensajes” en la página 986.

Manejadores de conexión compartidos en MQCONNX

Se pueden compartir manejadores entre distintas hebras del mismo proceso utilizando manejadores de conexión compartidos.

Cuando especifica un manejador de conexión compartido, el manejador de conexión devuelto por la llamada MQCONNX puede pasarse en las llamadas MQI posteriores efectuadas en cualquier hebra del proceso.

Nota: Puede utilizar un manejador de conexión compartido en un IBM MQ MQI client para conectarse a un gestor de colas de servidor que no dé soporte a los manejadores de conexión compartidos.

Para obtener más información, consulte “Utilización de MQCONNX” en la página 942.

Creación de aplicaciones para IBM MQ MQI clients

Se pueden crear y ejecutar aplicaciones en el entorno de IBM MQ MQI client. La aplicación se tiene que compilar y enlazar con el IBM MQ MQI client usado. La forma en que se compilan y enlazan las aplicaciones varía en función de la plataforma y del lenguaje de programación utilizado.

Si una aplicación se debe ejecutar en un entorno de cliente, puede escribirla en los lenguajes indicados en la tabla siguiente:

Tabla 133. Lenguajes de programación soportados en los entornos de cliente						
Plataforma de cliente	**C**	**C++**	**COBOL**	**pTAL**	**RPG**	**Visual Basic**
AIX	Sí	Sí	Sí			
IBM i	Sí		Sí		Sí	
Linux	Sí	Sí	Sí			
Windows	Sí	Sí	Sí			Sí

Multi

Enlace de aplicaciones C con el código IBM MQ MQI client

Una vez haya escrito la aplicación IBM MQ que desea ejecutar en IBM MQ MQI client, debe enlazarla al código IBM MQ MQI client.

Puede enlaza la aplicación al código IBM MQ MQI client de dos formas:

1. Directamente, conectando la aplicación con un gestor de colas, en cuyo caso este tendría que estar en la misma máquina que la aplicación.
2. Con un archivo de biblioteca de cliente, que da acceso a gestores de colas en la misma máquina o en otra distinta.

IBM MQ proporciona un archivo de biblioteca de cliente para cada entorno:

AIX **AIX**

La biblioteca libmqic.a para aplicaciones sin hebras o la biblioteca libmqic_r.a para aplicaciones con hebras.

Linux **Linux**

La biblioteca libmqic.so para aplicaciones sin hebras o biblioteca libmqic_r.so para aplicaciones con hebras.

IBM i **IBM i**

Enlace la aplicación cliente con el programa de servicio cliente LIBMQIC para aplicaciones sin hebras o con el programa de servicio LIBMQIC_R para aplicaciones con hebras.

Windows **Windows**

MQIC32.LIB.

ALW

Enlace de aplicaciones en C++ con el código IBM MQ MQI client

Puede escribir aplicaciones para ejecutarlas en el cliente en C++. Los métodos de compilación varían según el entorno.

Para obtener más información sobre cómo enlazar las aplicaciones en C++, consulte Creación de programas C++ de IBM MQ.

Para obtener detalles completos sobre todos los aspectos del uso de C++, consulte Utilización de C++

Multi

Enlace de aplicaciones COBOL con el código IBM MQ MQI client

Una vez que haya escrito una aplicación COBOL en la que desee ejecutar IBM MQ MQI client, debe enlazarla a una biblioteca apropiada.

IBM MQ proporciona un archivo de biblioteca de cliente para cada entorno:

AIX **AIX**

Enlace una aplicación COBOL sin hilos con la biblioteca libmqicb.a o una aplicación COBOL con hilos con libmqicb_r.a.

IBM i
Enlace la aplicación cliente con el programa de servicio AMQCSTUB para aplicaciones sin hebras o con el programa de servicio AMQCSTUB_R para aplicaciones con hebras.

Windows
Enlace el código de aplicación con la biblioteca MQICCBB para COBOL de 32 bits. El IBM MQ MQI client for Windows no soporta COBOL de 16 bits.

Windows *Enlace de aplicaciones de Visual Basic con el código de IBM MQ MQI client*

Puede enlazar aplicaciones de Microsoft Visual Basic con el código de IBM MQ MQI client en Windows.

Deprecated

A partir de IBM MQ 9.0, el soporte para Microsoft Visual Basic 6.0 está en desuso. Las clases de IBM MQ para .NET son la tecnología de sustitución recomendada. Para obtener más información, consulte Desarrollo de aplicaciones .NET.

Enlace la aplicación de Visual Basic con los archivos de inclusión siguientes:

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
mandatos PCF

CMQXB.bas
Canales

Establezca `mqtype=2` para el cliente en el compilador de Visual Basic para asegurar la selección automática correcta de la dll de cliente:

MQIC32.dll
Windows 7, Windows 8, Windows 2008 y Windows 2012

Conceptos relacionados

"Codificación en Visual Basic" en la página 1079
Información que se debe tener en cuenta al codificar programas de IBM MQ en Microsoft Visual Basic. Visual Basic solo está soportado en Windows.

"Preparación de programas Visual Basic en Windows" en la página 1045
Información a tener en cuenta cuando se utilizan programas de Microsoft Visual Basic en Windows.

Ejecución de aplicaciones en el entorno de IBM MQ MQI client

Puede ejecutar una aplicación de IBM MQ en un entorno de IBM MQ completo y en un entorno de IBM MQ MQI client sin cambiar el código, siempre que se cumplan determinadas condiciones.

Estas condiciones son que:

- La aplicación no necesite conectarse a más de un gestor de colas simultáneamente.
- El nombre de gestor de colas no tenga un asterisco (*) como prefijo en una llamada `MQCONN` o `MQCONNX`.
- La aplicación no tiene que utilizar ninguna de las excepciones listadas en ¿Qué aplicaciones se ejecutan en IBM MQ MQI client?

Nota: Las bibliotecas que utilice en el momento de la edición de enlaces determina el entorno en el que debe ejecutarse la aplicación.

Cuando trabaje en el entorno de IBM MQ MQI client, recuerde que:

- Cada aplicación que se ejecuta en el entorno de IBM MQ MQI client tiene sus propias conexiones con los servidores. Una aplicación establece una conexión con un servidor cada vez que emite una llamada `MQCONN` o `MQCONNX`.

- Una aplicación envía y obtiene mensajes de forma síncrona. Esto implica una espera entre el momento de emisión de la llamada en el cliente y la devolución de un código de terminación y de motivo a través de la red.
- Toda la conversión de datos la realiza el servidor, pero consulte también MQCCSID para obtener información sobre la alteración temporal del CCSID configurado de la máquina.

Conexión de aplicaciones de IBM MQ MQI client a gestores de colas

Una aplicación que se ejecuta en un entorno de IBM MQ MQI client se puede conectar a un gestor de colas de varias maneras. Puede utilizar variables de entorno, la estructura MQCNO o una tabla de definición de cliente.

Cuando una aplicación que se ejecuta en un entorno de cliente de IBM MQ emite una llamada MQCONN o MQCONNX, el cliente identifica cómo se debe realizar la conexión. Cuando una aplicación emite una llamada MQCONNX en un cliente de IBM MQ, la biblioteca de cliente MQI busca la información de canal de cliente en el orden siguiente:

1. Utilizando el contenido de los campos `ClientConnOffset` o `ClientConnPtr` de la estructura MQCNO (si se ha suministrado). Estos campos identifican la estructura de definición de canal (MQCD) para utilizarla como la definición del canal de conexión de cliente. Los detalles de la conexión pueden alterarse temporalmente utilizando una salida de preconexión. Para obtener más información, consulte “Referencia a las definiciones de conexión utilizando una salida de preconexión desde un depósito” en la página 1018.
2. Si se establece la variable de entorno **MQSERVER** , se utiliza el canal que define.
3. Si se define un archivo `mqclient.ini` y la stanza Channels contiene un atributo **ServerConnectionParms** , se utiliza el canal que define. Para obtener más información, consulte Archivo de configuración deIBM MQ MQI client , `mqclient.ini` y stanza Channels del archivo de configuración del cliente.
4. Si se establecen las variables de entorno **MQCHLLIB** y **MQCHLTAB** , se utiliza la tabla de definición de canal de cliente a la que apuntan. De forma alternativa, a partir de IBM MQ 9.0, la variable de entorno **MQCCDTURL** proporciona la capacidad equivalente para establecer una combinación de las variables de entorno **MQCHLLIB** y **MQCHLTAB** . Si se establece **MQCCDTURL** , se utiliza la tabla de definición de canal de cliente a la que apunta. Para obtener más información, consulte Acceso de URL deURL a la CCDT.
5. Si se define un archivo `mqclient.ini` y la stanza Channels contiene los atributos **ChannelDefinitionDirectory** y **ChannelDefinitionFile** , estos atributos se utilizan para localizar la tabla de definición de canal de cliente. Para obtener más información, consulte Archivo de configuración deIBM MQ MQI client , `mqclient.ini` y stanza Channels del archivo de configuración del cliente.
6. Por último, si las variables de entorno no están establecidas, el cliente busca una tabla de definición de canal de cliente con una vía de acceso y un nombre establecidos a partir del atributo **DefaultPrefix** de la stanza AllQueueManagers en el archivo `mqs.ini` . Para obtener más información, consulte AllQueueManagers stanza del archivo mqs.ini.

   Si la búsqueda de una tabla de definición de canal de cliente falla, el cliente utiliza las vías de acceso siguientes:

   - Linux AIX En AIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - Windows En Windows: `C:\Archivos de programa\IBM\MQ\amqclchl.tab`
   - IBM i En IBM i: `/QIBM/UserData/mqm/@ipcc`
   - MQ Appliance En IBM MQ Appliance: *QMname*_AMQCLCHL.TAB. Aparecen en `mqbackup:// URI`.

La primera de la opciones descritas en la lista anterior (utilizando los campos `ClientConnOffset` o `ClientConnPtr` de MQCNO) solo está soportada por la llamada MQCONNX. Si la aplicación utiliza MQCONN en lugar de MQCONNX, la información de canal se busca de las cinco formas restantes en el

orden que se muestra en la lista. Si el cliente no consigue encontrar la información de canal, la llamada MQCONN o MQCONNX falla.

El nombre de canal (para la conexión de cliente) debe coincidir con el nombre de canal de conexión de servidor para que la llamada MQCONN o MQCONNX se realice satisfactoriamente.

Conceptos relacionados

Acceso direccionable web a la tabla de definición de canal de cliente

Tareas relacionadas

Configurar conexiones entre el servidor y el cliente

Referencia relacionada

Tabla de definiciones de canal de cliente

MQCNO - Opciones de conexión

Conexión de aplicaciones cliente con gestores de colas utilizando variables de entorno

Se puede suministrar información de canal de cliente a una aplicación que se ejecuta en un entorno de cliente mediante variables de entorno.

Una aplicación que se ejecuta en un entorno de IBM MQ MQI client se puede conectar a un gestor de colas utilizando las variables de entorno siguientes:

MQSERVER
: La variable de entorno **MQSERVER** se utiliza para definir un canal mínimo. **MQSERVER** especifica la ubicación del servidor IBM MQ y el método de comunicación que se va a utilizar.

MQCHLLIB
: La variable de entorno **MQCHLLIB** especifica la vía de acceso del directorio al archivo que contiene la tabla de definición de canal de cliente (CCDT). El archivo se crea en el servidor, pero se puede copiar en la estación de trabajo de IBM MQ MQI client.

MQCHLTAB
: La variable de entorno **MQCHLTAB** especifica el nombre del archivo que contiene la tabla de definición de canal de cliente (CCDT).

A partir de IBM MQ 9.0, la variable de entorno **MQCCDTURL** proporciona la capacidad equivalente para establecer una combinación de las variables de entorno **MQCHLLIB** y **MQCHLTAB** . **MQCCDTURL** le permite proporcionar un archivo, ftp o http URL como un único valor del que se puede obtener una tabla de definición de canal de cliente. Para obtener más información, consulte Acceso direccionable web a la tabla de definiciones de canales de cliente.

Conexión de aplicaciones cliente con gestores de colas utilizando la estructura MQCNO

Se puede especificar la definición del canal en una estructura de definición de canal (MQCD), que se proporciona utilizando la estructura MQCNO de la llamada MQCONNX.

Para obtener más información, consulte Creación de un canal de conexión de cliente en IBM MQ MQI client utilizando MQCNO.

Conexión de aplicaciones cliente con gestores de colas utilizando una tabla de definiciones de canal de cliente

Si se utiliza el comando MQSC DEFINE CHANNEL, los detalles que se proporcionan se colocan en la tabla de definiciones de canal de cliente (CCDT). El contenido del parámetro **QMgrName** de la llamada MQCONN o MQCONNX determina el gestor de colas con el que se conecta el cliente.

El cliente accede a este archivo para determinar el canal que va a usar una aplicación. Donde hay más de una definición de canal adecuada, la elección del canal se ve influida por los atributos de ponderación de canal cliente (CLNTWGHT) y de afinidad de conexión (AFFINITY).

Utilización de la reconexión de cliente automática

Puede hacer que las aplicaciones cliente se reconecten automáticamente, sin tener que escribir código adicional, configurando una serie de componentes.

La reconexión de cliente automática es *en línea*. La conexión se restaura automáticamente en cualquier punto del programa de aplicación cliente y se restauran todos los manejadores para abrir objetos.

Por el contrario, la reconexión manual necesita que la aplicación cliente vuelva a crear una conexión mediante MQCONN o MQCONNX y que vuelva a abrir los objetos. La reconexión de cliente automática es adecuada para muchas aplicaciones cliente, pero no para todas.

Para obtener más información, consulte Reconexión de cliente automática.

Función de la tabla de definiciones de canales de cliente
La tabla de definiciones de canales de cliente (CCDT) contiene definiciones de los canales de conexión de cliente. La tabla es especialmente útil si las aplicaciones cliente pueden necesitar conectar con varios gestores de colas alternativos.

La tabla de definiciones de canales de cliente se crea cuando define un gestor de colas. El mismo archivo puede ser utilizado por más de un cliente de IBM MQ.

Existen varias formas de que una aplicación cliente utilice una CCDT. La CCDT se puede copiar en el sistema cliente. Puede copiar la CCDT en una ubicación compartida por más de un cliente. Puede hacer que la CCDT sea accesible para el cliente como un archivo compartido, mientras sigue ubicada en el servidor.

Desde IBM MQ 9.0, la CCDT se puede alojar en una ubicación central que es accesible a través de un URI, eliminando la necesidad de actualizar individualmente la CCDT para cada cliente desplegado.

Conceptos relacionados
Acceso direccionable web a la tabla de definición de canal de cliente
Tareas relacionadas
Acceso a las definiciones de canal de conexión de cliente
Referencia relacionada
Tabla de definiciones de canal de cliente

Grupos de gestores de colas en la CCDT
Se puede definir un conjunto de conexiones en la tabla de definiciones de canal de cliente (CCDT) como un *grupo de gestores de colas*. Se puede conectar una aplicación con un gestor de colas que forme parte de un grupo de gestores de colas. Para ello, prefije el nombre del gestor de colas a una una llamada MQCONN o MQCONNX con un asterisco.

Podría optarse por definir conexiones con más de una máquina servidora porque:

- Se desea conectar un cliente con cualquiera de los gestores del conjunto de gestores de colas que está ejecutando, para mejorar la disponibilidad.
- Se desea volver a conectar un cliente con el mismo gestor de colas con el que se conectó satisfactoriamente la última vez, pero conectarse con un gestor de colas diferente si falla la conexión.
- Se desea poder reintentar una conexión cliente con un gestor de colas diferente si falla la conexión, emitiendo MQCONN de nuevo en el programa cliente.
- Se desea reconectar automáticamente una conexión cliente con otro gestor de colas si falla la conexión sin escribir ningún código de cliente.
- Se desea reconectar automáticamente una conexión cliente con una instancia distinta de un gestor de colas de varias instancias si toma el control una instancia en espera sin escribir ningún código de cliente.
- Se desea distribuir las conexiones de cliente entre varios gestores de colas, con más clientes conectándose con unos gestores de colas que con otros.
- Se desea distribuir la reconexión de muchas conexiones cliente entre varios gestores de cola y a lo largo del tiempo, en caso que un volumen elevado de conexiones provocara un fallo.
- Se desea poder trasladar los gestores de colas sin necesidad de cambiar ningún código de aplicación cliente.
- Se desea escribir programas de aplicación cliente que no necesiten conocer nombres de gestores de colas.

No siempre resulta adecuado conectarse con distintos gestores de colas. Un cliente transaccional ampliado o un cliente Java en WebSphere Application Server, por ejemplo, podría tener que conectarse a una instancia de gestor de colas predecible. La reconexión automática de cliente no está soportada en IBM MQ classes for Java.

Un grupo de gestores de colas es un conjunto de conexiones definidas en la tabla de definiciones de canal de cliente (CCDT). El conjunto se define por los miembros que tienen el mismo valor de atributo **QMNAME** en sus definiciones de canal.

Figura 97 en la página 948 es una representación gráfica de una tabla de conexiones de cliente, que muestra tres grupos de gestores de colas, dos grupos de gestores de colas denominados en la CCDT **QMNAME** (QM1) y **QMNAME** (QMGrp1), y un grupo en blanco o predeterminado que aparece como **QMNAME** ('').

1. El grupo de gestores de colas QM1 tiene tres canales de conexiones de cliente que lo conectan con los gestores de colas QM1 y QM2. QM1 podría ser un gestor de colas multiinstancia ubicado en dos servidores distintos.
2. El grupo de gestores de colas predeterminado tiene seis canales de conexión de cliente que lo conectan con todos los gestores de colas.
3. QMGrp1 tiene canales de conexión de cliente con dos gestores de colas, QM4 y QM5.

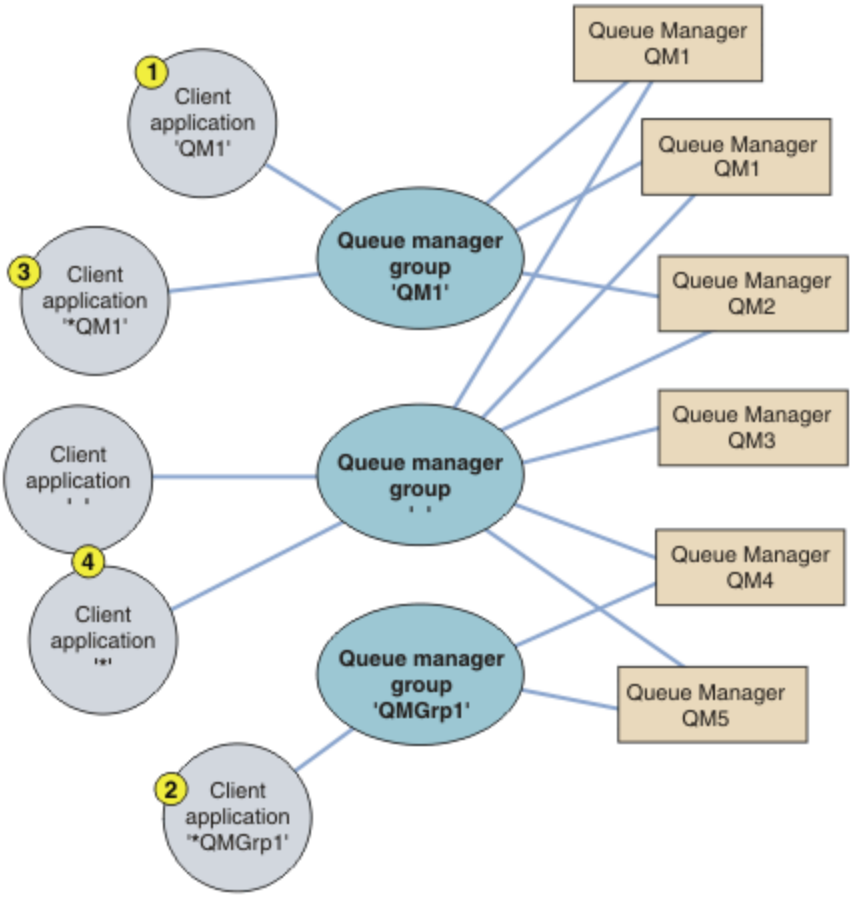


Figura 97. Grupos de gestores de colas

Se describen cuatro ejemplos distintos de uso de esta tabla de conexiones de cliente con la ayuda de las aplicaciones cliente numeradas en Figura 97 en la página 948.

1. En el primer ejemplo, la aplicación cliente pasa un nombre de gestor de colas, QM1, como parámetro **QmgrName** a su llamada MQI MQCONN o MQCONNX . El código de cliente IBM MQ selecciona el grupo de gestores de colas coincidente, QM1. El grupo contiene tres canales de conexión, e IBM MQ MQI

client intenta conectarse a QM1 utilizando cada uno de estos canales por turnos hasta que encuentra un escucha IBM MQ para la conexión conectada a un gestor de colas en ejecución denominado QM1.

El orden de intentos de conexión depende del valor del atributo AFFINITY de la conexión con el cliente y de las ponderaciones de los canales cliente. Dentro de estas limitaciones, el orden de intentos de conexión es aleatorio, entre las tres conexiones posibles y a lo largo del tiempo, para distribuir la carga del establecimiento de conexiones.

La llamada MQCONN o MQCONNX emitida por la aplicación cliente tiene éxito cuando se establece una conexión con una instancia de QM1 que está ejecutando.

2. En el segundo ejemplo, la aplicación cliente pasa un nombre de gestor de colas con un asterisco, *QMGrp1 como parámetro **QmgrName** a su llamada MQI MQCONN o MQCONNX . El cliente IBM MQ selecciona el grupo de gestores de coles coincidente, QMGrp1. Este grupo contiene dos canales de conexión de cliente y el IBM MQ MQI client intenta conectarse con *cualquier* gestor de colas utilizando cada canal por turnos. En este ejemplo, el IBM MQ MQI client tiene que realizar una conexión satisfactoria; el nombre del gestor de colas con el que se conecta no importa.

   La regla que determina el orden de los intentos de conexión es la misma que antes. La única diferencia es que, al preceder el nombre del gestor de colas con un asterisco, el cliente indica que el nombre del gestor de colas no es relevante.

   Las llamadas MQCONN o MQCONNX emitidas por la aplicación cliente tienen éxito cuando se establece una conexión con una instancia en ejecución de cualquier gestor de colas conectado mediante los canales en el grupo de gestores de colas QMGrp1.

3. El tercer ejemplo es esencialmente el mismo que el segundo porque el parámetro **QmgrName** tiene como prefijo un asterisco, *QM1. En el ejemplo se ilustra que no es posible determinar con qué gestor de colas va a conectarse una conexión de canal cliente a partir del atributo QMNAME en una definición de canal. El hecho de que el atributo **QMNAME** de la definición de canal sea QM1 no basta para obligar a que se establezca una conexión con un gestor de colas llamado QM1. Si la aplicación cliente prefija su parámetro **QmgrName** con un asterisco, cualquier gestor de colas es un posible destino de conexión.

   En este caso, las llamadas MQCONN o MQCONNX emitidas por la aplicación cliente serán satisfactorias cuando se establezca una conexión con una instancia en ejecución de QM1 o de QM2.

4. El cuarto ejemplo ilustra la utilización del grupo predeterminado. En este caso, la aplicación cliente pasa un asterisco, '*', o en blanco '', como el parámetro **QmgrName** a su llamada MQI MQCONN o MQCONNX . Por convenio en la definición de canal de cliente, un atributo **QMNAME** en blanco significa el grupo de gestores de colas predeterminado y un parámetro **QmgrName** en blanco o asterisco coincide con un atributo **QMNAME** en blanco.

   En este ejemplo, el grupo de gestores de colas predeterminado tiene conexiones de canal cliente con todos los gestores de colas. Seleccionando el grupo de gestores de colas predeterminado, la aplicación puede conectarse con cualquier gestor de colas del grupo.

   Las llamadas MQCONN o MQCONNX emitidas por la aplicación cliente tienen éxito cuando se establece una conexión con una instancia en ejecución de cualquier gestor de colas.

   Nota: El grupo predeterminado es diferente de un gestor de colas predeterminado, aunque una aplicación utiliza un parámetro **QmgrName** en blanco para conectarse al grupo de gestores de colas predeterminado o al gestor de colas predeterminado. El concepto de grupo de gestores de colas predeterminado solo es relevante para una aplicación cliente y el gestor de colas predeterminado lo es para una aplicación de servidor.

Defina los canales de conexiones cliente en un solo gestor de colas, incluyendo los canales que se conectan con un segundo o con un tercer gestor de colas. No los defina en dos gestores de colas y luego intente fusionar ambas tablas de definiciones de canal cliente. El cliente solo puede acceder a una única tabla de definiciones de canal de cliente.

Ejemplos

Vuelva a consultar la lista de razones para utilizar los grupos de gestores de colas al principio de este tema. ¿Cómo ofrece esas capacidades el utilizar un grupo de gestores de colas?

Conexión con cualquier conjunto de gestores de colas.
Defina un grupo de gestores de colas con conexiones a todos los gestores de colas del conjunto y conéctese al grupo utilizando el parámetro **QmgrName** con un asterisco como prefijo.

Reconexión con el mismo gestor de colas, pero conéctese con uno distinto si el gestor de colas con el que se conectó por última vez no está disponible.
Defina un grupo de gestores de colas como antes, pero defina el atributo **AFFINITY**(PREFERRED) en cada definición de canal de cliente.

Reintento de una conexión con otro gestor de colas si falla una conexión.
Conexión con un grupo de gestores de colas y vuelva a emitir las llamadas MQI MQCONN o MQCONNX si se interrumpe la conexión o falla el gestor de colas.

Reconexión automática con otro gestor de colas si falla una conexión.
Conéctese con un grupo de gestores de colas utilizando la opción MQCONNX **MQCNO** MQCNO_RECONNECT.

Reconexión automática con a una instancia diferente de un gestor de colas con varias instancias.
Haga lo mismo que en el ejemplo anterior. En este caso, si desea restringir el grupo de gestores de colas para conectarse con las instancias de un gestor de colas multiinstancia concreto, defina el grupo con conexiones únicamente a las instancias del gestor de colas multiinstancia.

También puede solicitar a la aplicación cliente que emita su llamada MQI MQCONN o MQCONNX sin ningún asterisco con el prefijo del parámetro **QmgrName** . De ese modo, la aplicación cliente solo podrá conectarse con el gestor de colas indicado. Por último, puede establecer la opción **MQCNO** a MQCNO_RECONNECT_Q_MGR. Esta opción acepta reconexiones con el mismo gestor de colas con el que estaba conectado previamente. También puede utilizar este valor para restringir las reconexiones a la misma instancia de un gestor de colas normal.

Distribución de conexiones cliente entre gestores de colas, con más clientes conectados con algunos gestores de colas que con otros.
Defina un grupo de gestores de colas y establezca el atributo **CLNTWGHT** en cada definiciones de canal de cliente para distribuir las conexiones de forma irregular.

Reparto de la carga de reconexiones de cliente de forma irregular y a lo largo del tiempo, tras un fallo de conexión o del gestor de colas.
Haga lo mismo que en el ejemplo anterior. El IBM MQ MQI client distribuye de forma aleatoria las reconexiones entre los gestores de colas y reparte las reconexiones a lo largo del tiempo.

Desplazamiento de los gestores de colas sin cambiar ningún código cliente.
La CCDT desacopla una aplicación cliente de la ubicación del gestor de colas. La CCDT es un archivo de datos que se puede definir en el cliente, leer desde una ubicación compartida o recuperar de un servidor web. Para obtener más información, consulte Tabla de definiciones de canal cliente.

Desarrollo de una aplicación cliente que no conozca los nombres de los gestores de colas.
Utilice los nombres de grupos de gestores de colas y establezca una convención de nombres e grupos de gestores de colas que sea relevante para las aplicaciones cliente de la organización y que refleje la arquitectura de las soluciones y no la del nombrado de los gestores de colas.

z/OS *Conexión a los grupos de compartición de colas*

Puede conectar la aplicación a un gestor de colas que forme parte de un grupo de compartición de colas. Esto se puede realizar utilizando el nombre del grupo de compartición de colas, en lugar del nombre del gestor de colas en la llamada MQCONN o MQCONNX.

Los grupos de compartición de colas tienen un nombre de hasta cuatro caracteres. El nombre debe ser exclusivo en la red y debe ser diferente de los nombres de gestor de colas.

La definición de canal de cliente debe utilizar la interfaz genérica del grupo de compartición de colas para conectarse a un gestor de colas disponible en el grupo. Si desea más información, consulte Conexión de un cliente a un grupo de compartición de colas. Se realiza una comprobación para asegurarse de que el gestor de colas al que se conecta el escucha es miembro del grupo de compartición de colas.

Si desea más información sobre colas compartidas, consulte Colas compartidas y grupos de compartición de colas.

Ejemplos de ponderación y afinidad de canal
Estos ejemplos ilustran cómo se seleccionan los canales de conexión de cliente cuando se utilizan `ClientChannelWeights` (ponderacions de canal cliente) distintas de cero.

Los atributos de canal `ClientChannelWeight` y `ConnectionAffinity` controlan cómo se seleccionan los canales de conexión de cliente cuando hay más de un canal adecuado para una conexión. Estos canales están configurados para conectarse con diferentes gestores de colas a fin de proporcionar una disponibilidad mayor, un mayor equilibro de carga o ambos. Las llamadas MQCONN que podrían dar como resultado una conexión con uno de varios gestores de colas deben anteponer al nombre del gestor de colas un asterisco tal como se describe en: Ejemplos de llamadas MQCONN: Ejemplo 1. El nombre del gestor de colas incluye un asterisco (*).

Los canales candidatos aplicables para una conexión son aquellos en los que el atributo QMNAME coincide con el nombre del gestor de colas especificado en la llamada MQCONN. Si todos los canales aplicables para una conexión tienen un `ClientChannelPeso` de cero (el valor predeterminado), se seleccionan en orden alfabético como en el ejemplo: Ejemplos de llamadas MQCONN: Ejemplo 1. El nombre del gestor de colas incluye un asterisco (*).

Los ejemplos siguientes ilustran lo que sucede cuando se utilizan `ClientChannelWeights` distintas de cero. Observe que, puesto que esta característica implica una selección de canal pseudoaleatoria, los ejemplos muestran una secuencia de acciones que podría ocurrir, en lugar de lo que se producirá definitivamente.

Ejemplo 1. Selección de canales cuando ConnectionAffinity se establece en PREFERRED
Este ejemplo ilustra cómo IBM MQ MQI client selecciona un canal en un CCDT, donde ConnectionAffinity está establecido en PREFERRED.

En este ejemplo, una serie de máquinas cliente utilizan una tabla de definiciones de canal de cliente (CCDT) proporcionada por un gestor de colas. La CCDT incluye los canales de conexión cliente con los atributos siguientes (se muestran utilizando la sintaxis del comando DEFINE CHANNEL):

```
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)
```

La aplicación emite MQCONN(*CORE)

El canal A no es candidato para esta conexión, ya que el atributo QMNAME no coincide. Los canales B, C y D se identifican como candidatos y se colocan en un orden de preferencia basado en su ponderación. En este ejemplo, el orden puede ser C, B, D. El cliente intenta conectarse al gestor de colas en core2.ops.company.example. El nombre del gestor de colas en esa dirección no se comprueba, ya que la llamada MQCONN incluye un asterisco en el nombre de gestor de colas.

Es importante tener que cuenta que, con `AFFINITY(PREFERRED)`, cada vez que esta máquina cliente concreta se conecte, colocará los canales en el mismo orden inicial de preferencia. Esto se aplica incluso cuando las conexiones proceden de procesos diferentes o tienen lugar en momentos diferentes.

En este ejemplo, no se puede acceder al gestor de colas en core.2.ops.company.example. El cliente intenta conectarse con core1.ops.company.example porque el canal B es el siguiente en el orden de preferencia. Además, el canal C se ha degradado para convertirse en el de menos preferencia.

La misma aplicación emite una segunda llamada MQCONN(*CORE). El canal C ha sido degradado por la conexión anterior, por lo que el canal más preferido es ahora B. Esta conexión se realiza con core1.ops.company.example.

Una segunda máquina que comparte la misma tabla de definiciones de canal de cliente coloca los canales en un orden inicial de preferencia distinto. Por ejemplo, D, B, C. En circunstancias normales, con todos los canales en funcionamiento, las aplicaciones de esta máquina se conectan a core3.ops.company.example mientras que las de la primera máquina se conectan a core2.ops.company.example. Esto permite repartir

la carga de trabajo de un gran número de clientes entre varios gestores de colas, al tiempo que cada cliente individual se conecta con el mismo gestor de colas si está disponible.

Ejemplo 2. Selección de canales cuando ConnectionAffinity se establece en NONE
Este ejemplo ilustra cómo un IBM MQ MQI client selecciona un canal de una CCDT, donde ConnectionAffinity está establecido en NONE.

En este ejemplo, una serie de clientes utilizan una tabla de definiciones de canal de cliente (CCDT) proporcionada por un gestor de colas. La CCDT incluye los canales de conexión cliente con los atributos siguientes (se muestran utilizando la sintaxis del comando DEFINE CHANNEL):

```
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)
```

La aplicación emite MQCONN(*CORE). Como en el ejemplo anterior, el canal A no se tiene en cuenta porque el atributo QMNAME no coincide. Se seleccionan los canales B, C o D en función de sus ponderaciones, con probabilidades de 50%, 30% o 20%. En este ejemplo, podría seleccionarse el canal B. No se ha creado ningún orden persistente de preferencia.

Se efectúa una segunda llamada MQCONN(*CORE). De nuevo, se selecciona uno de los tres canales aplicables, con las mismas posibilidades. En este ejemplo, se elige el canal C. Sin embargo, core2.ops.company.example no responde, por lo que se hace otra elección entre los canales candidatos restantes. Se selecciona el canal B y la aplicación se conecta con core1.ops.company.example.

Con AFFINITY(NONE), cada llamada MQCONN es independiente de las demás. Por lo tanto, cuando la aplicación de este ejemplo realice una tercera llamada MQCONN(*CORE), podría de nuevo intentar conectarse a través del canal interrumpido C, antes de elegir el B o el D.

Ejemplos de llamadas MQCONN
Ejemplos de utilización de MQCONN para conectarse con un gestor de colas determinado o con uno de un grupo de gestores de colas.

En cada uno de los ejemplos siguientes, la red es la misma; hay una conexión definida con dos servidores desde el mismo IBM MQ MQI client. (En estos ejemplos, podría utilizarse la llamada MQCONNX en lugar de la llamada MQCONN).

Hay dos gestores de colas que ejecutan en las máquinas servidoras, uno llamado SALE y el otro llamado SALE_BACKUP.

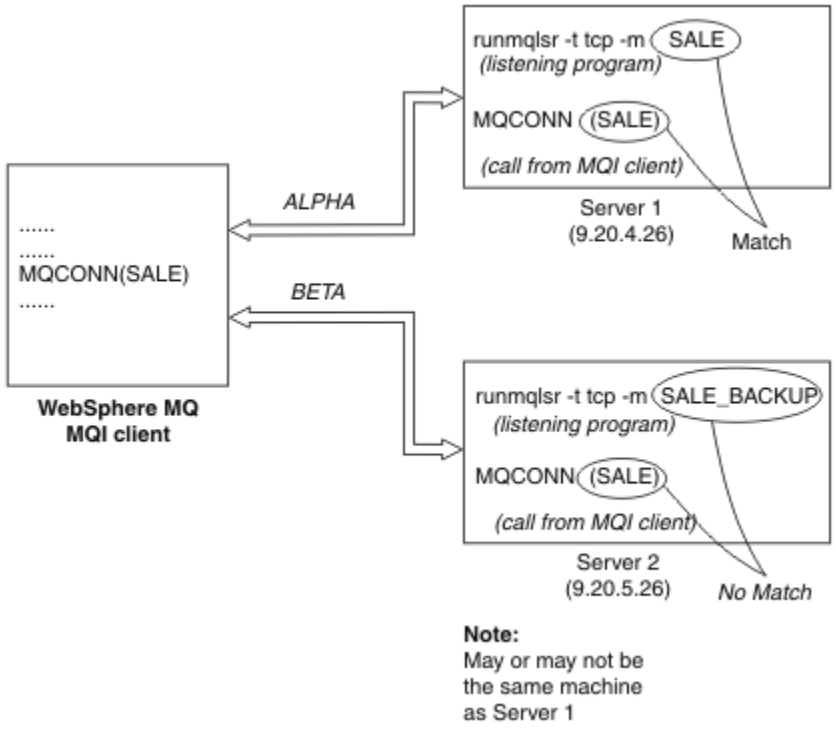


Figura 98. Ejemplo de MQCONN

Las definiciones de los canales de estos ejemplos son las siguientes:

Definiciones de SALE:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definición de SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

Las definiciones de canal de cliente pueden resumirse como sigue:

Nombre	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

Qué muestran los ejemplos de MQCONN
Los ejemplos muestran el uso de varios gestores de colas como un sistema de copia de seguridad.

Suponga que el enlace de comunicaciones con el Servidor 1 se interrumpe temporalmente. Se muestra el uso de varios gestores de colas como sistema de copia de seguridad.

Cada ejemplo cubre una llamada MQCONN distinta y proporciona una explicación de lo que sucede en el ejemplo específico presentado, aplicando las reglas siguientes:

1. La tabla de definiciones de canal de cliente (CCDT) se recorre en orden alfabético de nombre de canal en busca de un nombre de gestor de colas (campo QMNAME) que se corresponda con el proporcionado en la llamada MQCONN.
2. Si se encuentra una coincidencia, se utilizará la definición de canal.
3. Se intenta iniciar el canal de la máquina identificada por el nombre de conexión (CONNAME). Si esto se realiza satisfactoriamente, la aplicación continúa. Requiere:
   - Un escucha que ejecute en el servidor.
   - El escucha tiene que estar conectado con mismo gestor de colas con el que el cliente desea conectarse (si se ha especificado).
4. Si el falla el intento de iniciar el canal y hay más de una entrada en la tabla de definiciones de canal de cliente (en este ejemplo hay dos entradas), se buscará otra coincidencia en el archivo. Si se encuentra una coincidencia, el proceso continúa en el paso 1.
5. Si no se encuentra ninguna coincidencia o no quedan más entradas en la tabla de definiciones de canal de cliente y el canal no ha podido iniciarse, la aplicación no podrá conectarse. La llamada MQCONN devuelve los correspondientes códigos de terminación y razón. La aplicación puede tomar las medidas oportunas a partir de los códigos de razón y de terminación devueltos.

Ejemplo 1. El nombre del gestor de colas incluye un asterisco (*)
En este ejemplo, a la aplicación no le afecta el gestor de colas al que se conecta. La aplicación emite una llamada MQCONN para un nombre de gestor de colas que incluye un asterisco. Se elige un canal adecuado.

La aplicación emite:

```
MQCONN (*SALE)
```

Siguiendo las reglas, esto es lo que sucede en este ejemplo:

1. Se explora la tabla de definiciones de canal de cliente (CCDT) en busca del nombre de gestor de colas SALE, que coincide con la llamada MQCONN de la aplicación.
2. Se encuentran definiciones de canal de ALPHA y BETA.
3. Si un canal tiene un valor CLNTWGHT de 0, se selecciona este canal. Si ambos tienen un valor CLNTWGHT de 0, se selecciona el canal ALPHA, porque es el primero por orden alfabético. Si ambos canales tienen un valor CLNTWGHT distinto de cero, se selecciona un canal de forma aleatoria, en función de su ponderación.
4. Se intenta iniciar el canal.
5. Si se ha seleccionado el canal BETA, el intento de iniciarlo es satisfactorio.
6. Si se ha seleccionado el canal ALPHA, el intento de iniciarlo NO es satisfactorio, porque se interrumpe el enlace de comunicaciones. En tal caso, se siguen estos pasos:
   a. El único canal restante para el nombre de gestor de colas SALE es BETA.
   b. Se intenta iniciar este canal, de forma satisfactoria.
7. La comprobación efectuada para averiguar si está ejecutando un escucha indica que hay uno en ejecución. Este no está conectado con el gestor de colas SALE, pero como el parámetro de la llamada MQI contiene un asterisco (*), no se realiza ninguna comprobación. La aplicación se conecta con gestor de colas SALE_BACKUP y sigue procesando.

Ejemplo 2. Nombre de gestor de colas especificado
En este ejemplo, la aplicación tiene que conectarse con un gestor de colas determinado. La aplicación emite una llamada MQCONN para ese nombre de gestor de colas. Se elige un canal adecuado.

La aplicación requiere una conexión con un gestor de colas concreto llamado SALE, tal como se ve en la llamada MQI:

```
MQCONN (SALE)
```

Siguiendo las reglas, esto es lo que sucede en este ejemplo:

1. Se explora la tabla de definiciones de canal de cliente (CCDT), de forma secuencial por orden alfabético de nombres de canal, en busca del nombre de gestor de colas SALE, que coincide con la llamada MQCONN de la aplicación.
2. La primera definición de canal encontrada coincidente ALPHA.
3. Se ha intentado iniciar el canal; no se ha podido porque el enlace de comunicación está roto.
4. Se vuelve a explorar la tabla de definiciones de canal de cliente para buscar el nombre de gestor de colas SALE y se encuentra el nombre de canal BETA.
5. Se intenta iniciar el canal, esta vez de forma satisfactoria.
6. Una comprobación para ver si está ejecutando un escucha muestra que hay uno ejecutando, pero no está conectado con el gestor de colas SALE.
7. No hay más entradas en la tabla de definiciones de canal de cliente. La aplicación no puede continuar y recibe un código de retorno MQRC_Q_MGR_NOT_AVAILABLE.

Ejemplo 3. El nombre del gestor de colas está en blanco o es un asterisco (*)
En este ejemplo, a la aplicación no le afecta el gestor de colas al que se conecta. La aplicación emite un MQCONN que especifica un nombre de gestor de colas en blanco o un asterisco. Se elige un canal adecuado.

Se trata del mismo modo que se describe en "Ejemplo 1. El nombre del gestor de colas incluye un asterisco (*)" en la página 954.

Nota: Si esta aplicación se estaba ejecutando en un entorno distinto de un IBM MQ MQI client, y el nombre estaba en blanco, estaría intentando conectarse al gestor de colas predeterminado. Este no es el caso cuando se ejecuta desde un entorno de cliente; el gestor de colas al que se accede es el que está asociado con el escucha al que se conecta el canal.

La aplicación emite:

```
MQCONN ("")
```

o

```
MQCONN (*)
```

Siguiendo las reglas, esto es lo que sucede en este ejemplo:

1. La tabla de definición de canal de cliente (CCDT) se explora en secuencia alfabética de nombre de canal, para un nombre de gestor de colas que está en blanco, que coincide con la llamada MQCONN de la aplicación.
2. La entrada del nombre de canal ALPHA tiene un nombre de gestor de colas en la definición de SALE. Este nombre no coincide con el parámetro de la llamada MQCONN, que requiere que el nombre de gestor de colas esté en blanco.
3. La siguiente entrada corresponde al nombre de canal BETA.
4. El `queue manager name` en la definición es SALE. De nuevo, el nombre no coincide con el parámetro de la llamada MQCONN, que requiere que el nombre del gestor de colas esté en blanco.
5. No hay más entradas en la tabla de definiciones de canal de cliente. La aplicación no puede continuar y recibe un código de retorno MQRC_Q_MGR_NOT_AVAILABLE.

Desencadenamiento en un entorno cliente

Los mensajes enviados por las aplicaciones IBM MQ que se ejecutan en IBM MQ MQI clients contribuyen al desencadenamiento exactamente de la misma forma que cualquier otro mensaje, y se pueden utilizar para desencadenar programas tanto en el servidor, como en el cliente.

El desencadenamiento se explica en detalle en Canales de desencadenamiento.

El supervisor desencadenante y la aplicación que va a iniciarse han de estar en el mismo sistema.

Las características predeterminadas de la cola desencadenada son las mismas que las del entorno de servidor. En concreto, si no se especifica ninguna opción de control de punto de sincronización MQPMO en una aplicación cliente que coloca mensajes en una cola desencadenada que es local a un gestor de colas z/OS, los mensajes se colocan dentro de una unidad de trabajo. Si se cumple la condición de desencadenamiento, el mensaje desencadenante se coloca en la cola de inicio dentro de la misma unidad de trabajo y no puede ser recuperado por el supervisor desencadenante hasta que finaliza la unidad de trabajo. El proceso que se va a desencadenar no se inicia hasta que finaliza la unidad de trabajo.

Definición de proceso

Un proceso se define en el servidor, ya que se asocia a la cola que tiene configurado el desencadenamiento.

El objeto de proceso define lo que tiene que desencadenarse. Si el cliente y el servidor no se ejecutan en la misma plataforma, los procesos iniciados por el supervisor desencadenante deben definir *ApplType*; de lo contrario, el servidor toma sus definiciones predeterminadas (es decir, el tipo de aplicación que normalmente está asociada con la máquina del servidor) y provoca una anomalía.

Por ejemplo, si el supervisor desencadenante se ejecuta en un IBM MQ MQI client y desea enviar una solicitud a un servidor en otro sistema operativo, MQAT_WINDOWS_NT debe estar definido; de lo contrario, el otro sistema operativo utiliza sus definiciones predeterminadas y el proceso falla.

Supervisor desencadenante

El supervisor desencadenante proporcionado por productos que no son z/OS IBM MQ se ejecuta en los entornos de clientes para sistemas IBM i IBM i, AIX, Linux, and Windows.

Para ejecutar el supervisor desencadenante, emita uno de estos comandos:

- IBM i En IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW En plataformas AIX, Linux, and Windows :

```
runmqtmc [-m QMgrName] [-q InitQ]
```

La cola de inicio predeterminada es SYSTEM.DEFAULT.INITIATION.QUEUE en el gestor de colas predeterminado. La cola de inicio es donde el supervisor desencadenante busca los mensajes desencadenantes. A continuación, invoca programas para los correspondientes mensajes desencadenantes. Este supervisor desencadenante soporta al tipo de aplicación predeterminado y es el mismo que `runmqtrm`, salvo que enlaza las bibliotecas de cliente.

La cadena de comandos, creada por el supervisor desencadenante, es la siguiente:

1. El *ApplicId* de la definición de proceso relevante. *ApplicId* es el nombre del programa que se va a ejecutar, tal como se especificaría en la línea de mandatos.
2. La estructura MQTMC2 que se obtiene de la cola de inicio, entrecomillada. Se inicia una cadena de comando que tiene esta cadena, tal y como se proporciona, entre comillas, para que el comando del sistema lo acepte como parámetro.
3. El *EnvrData* de la definición de proceso relevante.

El supervisor desencadenante no mira si hay otro mensaje en la cola de inicio mientras no termina la aplicación que ha iniciado. Si la aplicación tiene mucho procesamiento por hacer, puede que lleguen tantos mensajes que el supervisor desencadenante no de abasto. Hay dos maneras de hacer frente a esta situación:

1. Tener más supervisores desencadenantes en ejecución

   Si opta por tener más supervisores desencadenantes en ejecución, podrá controlar el número máximo de aplicaciones que se pueden ejecutar en cualquier momento.

2. Ejecutar las aplicaciones iniciadas en segundo plano

   Si opta por ejecutar aplicaciones en segundo plano, IBM MQ no impone ninguna restricción sobre el número de aplicaciones que se pueden ejecutar.

Para ejecutar la aplicación iniciada en segundo plano en sistemas AIX and Linux , debe colocar un & (ampersand) al final del *EnvrData* de la definición de proceso.

Aplicaciones CICS (no z/OS)
Un programa de aplicación no z/OS CICS que emite una llamada MQCONN o MQCONNX debe definirse en CEDA como RESIDENT. Si se vuelve a enlazar la aplicación de servidor CICS como cliente, se corre el riesgo de perder el soporte de punto de sincronización.

Un programa de aplicación no z/OS CICS que emite una llamada MQCONN o MQCONNX debe definirse en CEDA como RESIDENT. Para que el código residente sea lo más reducido posible, se puede enlazar con un programa aparte para emitir la llamada MQCONN o MQCONNX.

Si la variable de entorno MQSERVER se utiliza para definir la conexión de cliente, se debe especificar en el archivo CICSENV.CMD.

Las aplicaciones IBM MQ se pueden ejecutar en un entorno de servidor IBM MQ o en un cliente IBM MQ sin cambiar el código. Sin embargo, en un entorno de servidor IBM MQ, CICS puede actuar como un coordinador de punto de sincronización, y se utiliza EXEC CICS SYNCPOINT y EXEC CICS SYNCPOINT ROLLBACK, en lugar de **MQCMIT** y **MQBACK**. Si una aplicación CICS simplemente se vuelve a enlazar como cliente, se perderá el soporte de punto de sincronización. **MQCMIT** y **MQBACK** tienen que usarse para la aplicación que ejecuta en IBM MQ MQI client.

ALW Preparación y ejecución de las aplicaciones CICS y Tuxedo

Para ejecutar aplicaciones CICS y Tuxedo como aplicaciones cliente, utilice bibliotecas diferentes a las que utiliza con las aplicaciones de servidor. El ID de usuario con el que se ejecuta la aplicación también es diferente.

Para preparar las aplicaciones cliente CICS y Tuxedo de modo que se ejecuten como aplicaciones de IBM MQ MQI client, siga las instrucciones de la sección Configuración de un cliente de transacciones ampliado.

No obstante, tenga en cuenta que la información trata específicamente sobre la preparación de las aplicaciones CICS y Tuxedo, incluidos los programas de ejemplo proporcionados con IBM MQ y presupone que está preparando las aplicaciones para que se ejecuten en un sistema de servidor de IBM MQ. Por lo tanto, la información solo hace referencia a las bibliotecas de IBM MQ diseñadas para ser utilizadas en un sistema de servidor. Cuando prepare las aplicaciones cliente, debe realizar las acciones siguientes:

- Utilice la biblioteca de sistema cliente adecuada para los enlaces de lenguaje que utiliza su aplicación. Por ejemplo:
  - Linux AIX Para las aplicaciones escritas en C en AIX and Linux, utilice la biblioteca libmqic, en lugar de libmqm.
  - Windows En los sistemas Windows, utilice la biblioteca mqic.lib, en lugar de mqm.lib.

- En lugar de las bibliotecas que se muestran en la Tabla 134 en la página 958 y la Tabla 135 en la página 958, utilice las bibliotecas del sistema cliente adecuadas. Si en estas tablas no se lista una biblioteca del sistema servidor, utilice la misma biblioteca en un sistema cliente.

Tabla 134. Bibliotecas del sistema cliente en AIX and Linux

Biblioteca para un sistema servidor de IBM MQ	**Biblioteca equivalente para utilizar en un sistema cliente de IBM MQ**
libmqmxa	libmqcxa

Tabla 135. Bibliotecas del sistema cliente en sistemas Windows

Biblioteca para un sistema servidor de IBM MQ	**Biblioteca equivalente para utilizar en un sistema cliente de IBM MQ**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

El ID de usuario utilizado por una aplicación cliente

Cuando ejecuta una aplicación de servidor de IBM MQ en CICS, normalmente cambia del usuario de CICS al ID de usuario de la transacción. No obstante, cuando ejecuta una aplicación IBM MQ MQI client bajo CICS, retiene la autorización privilegiada de CICS.

ALW *Programas de ejemplo CICS y Tuxedo*

Programas de ejemplo CICS y Tuxedo para su uso en sistemas AIX, Linux, and Windows .

Tabla 136 en la página 958 muestra una lista de los programas de ejemplo CICS y Tuxedo que se proporcionan para su uso en sistemas cliente AIX and Linux. Tabla 137 en la página 958 muestra una lista de información equivalente para los sistemas cliente Windows. Las tablas también muestran una lista de los archivos que se utilizan para preparar y ejecutar los programas. Para ver una descripción de los programas de ejemplo, consulte “Ejemplo de transacción CICS” en la página 1108 y “Utilización de los ejemplos de TUXEDO en AIX, Linux, and Windows” en la página 1153.

Tabla 136. Programas de ejemplo para sistemas cliente AIX and Linux

Descripción	**Origen**	**Módulo ejecutable**
Programa CICS	amqscic0.ccs	amqscicc
Archivo de cabecera para el programa CICS	amqscih0.h	-
Programa cliente Tuxedo para transferir mensajes	amqstxpx.c	-
Programa cliente Tuxedo para obtener mensajes	amqstxgx.c	-
Programa servidor Tuxedo para los dos programas cliente	amqstxsx.c	-
Archivo UBBCONFIG para los programas Tuxedo	ubbstxcx.cfg	-
Archivo de tabla de campo para programas Tuxedo	amqstxvx.flds	-
Archivo de descripción de vista para programas Tuxedo	amqstxvx.v	-

Tabla 137. Programas de ejemplo para sistemas cliente Windows

Descripción	**Origen**	**Módulo ejecutable**
Transacción CICS	amqscic0.ccs	amqscicc

Tabla 137. Programas de ejemplo para sistemas cliente Windows (continuación)		
Descripción	**Origen**	**Módulo ejecutable**
Archivo de cabecera para la transacción CICS	amqscih0.h	-
Programa cliente Tuxedo para transferir mensajes	amqstxpx.c	-
Programa cliente Tuxedo para obtener mensajes	amqstxgx.c	-
Programa servidor Tuxedo para los dos programas cliente	amqstxsx.c	-
Archivo UBBCONFIG para los programas Tuxedo	ubbstxcx.cfg	-
Archivo de tabla de campo para programas Tuxedo	amqstxvx.fld	-
Archivo de descripción de vista para programas Tuxedo	amqstxvx.v	-
Archivo makefile para los programas Tuxedo	amqstxmc.mak	-
Archivo ENVFILE para los programas Tuxedo	amqstxen.env	-

ALW *Mensaje de error AMQ5203, como se ha modificado para las aplicaciones CICS y Tuxedo*

Cuando ejecuta las aplicaciones CICS o Tuxedo que utilizan un cliente de transacciones ampliado, es posible que vea mensajes de diagnóstico estándar. Uno de estos mensajes se ha modificado para utilizarlo con el cliente de transacciones ampliado.

Los mensajes que puede ver en los archivos de registro de errores de IBM MQ se describen en la sección Mensajes de diagnóstico AMQ4000-9999. El mensaje AMQ5203 se ha modificado para su uso con un cliente de transacciones ampliado. El siguiente es el texto del mensaje modificado:

AMQ5203: Se ha producido un error al llamar a la interfaz XA.

Explicación

El número de error es &2, donde un valor de 1 indica que el valor de los distintivos suministrados &1 no es válido, 2 indica que se ha intentado utilizar bibliotecas con hebras y sin hebras en el mismo proceso, 3 indica que se ha producido un error en el nombre de gestor de colas proporcionado '&3', 4 indica que el ID del gestor de recursos &1 no es válido, 5 indica que se ha intentado utilizar un segundo gestor de colas con el nombre '&3' cuando ya estaba conectado otro gestor de colas, 6 indica que se ha invocado el gestor de transacciones cuando la aplicación no estaba conectada a un gestor de colas, 7 indica que se ha realizado una llamada XA cuando otra llamada estaba en curso, 8 indica que la serie xa_info '&4' de la llamada xa_open contiene un valor de parámetro no válido en el nombre del parámetro '&5' y 9 indica que en la serie xa_info '&4' de la llamada xa_open falta un parámetro necesario, siendo el nombre del parámetro '&5'.

Respuesta del usuario

Corrija el error y vuelva a intentar la operación.

Windows Preparación y ejecución de aplicaciones Microsoft Transaction Server

Para preparar una aplicación MTS para que se ejecute como una aplicación IBM MQ MQI client, siga estas instrucciones según corresponda para el entorno.

Para obtener información general sobre cómo desarrollar aplicaciones Microsoft Transaction Server (MTS) que acceden a recursos IBM MQ, consulte la sección sobre MTS en el IBM MQ Help Center.

Para preparar una aplicación MTS para que se ejecute como una aplicación IBM MQ MQI client, realice una de las acciones siguientes para cada componente de la aplicación:

- Si el componente utiliza los enlaces del lenguaje C en la MQI, siga las instrucciones de "Preparación de programas C en Windows" en la página 1042, pero enlace el componente con la biblioteca mqicxa.lib en vez de mqic.lib.
- Si el componente utiliza las clases C++ de IBM MQ, siga las instrucciones de "Compilación de programas C++ en Windows" en la página 567, pero enlace el componente a la biblioteca imqx23vn.lib, en lugar de imqc23vn.lib.
- Si el componente utiliza los enlaces de lenguaje Visual Basic para la MQI, siga las instrucciones que aparecen en "Preparación de programas Visual Basic en Windows" en la página 1045, pero cuando defina el proyecto Visual Basic, escriba `MqType=3` en el campo **Argumentos de compilación condicional**,

Preparación y ejecución de aplicaciones IBM MQ JMS

Puede ejecutar aplicaciones IBM MQ JMS en la modalidad de cliente con WebSphere Application Server como su gestor de transacciones. Es posible que vea determinados mensajes de aviso.

Para preparar y ejecutar aplicaciones IBM MQ JMS en la modalidada cliente, con WebSphere Application Server como su gestor de transacciones, siga las instrucciones de "Utilización de IBM MQ classes for JMS/Jakarta Messaging" en la página 85.

Al ejecutar una aplicación cliente de IBM MQ JMS, es posible que vea los mensajes de aviso siguientes:

MQJE080
Unidades de licencia insuficientes - ejecute setmqcap

MQJE081
El archivo que contiene la información de unidades de licencia tiene un formato erróneo - ejecute setmqcap

MQJE082
No se ha podido encontrar el archivo que contiene la información de unidades de licencia - ejecute setmqcap.

Salidas de usuario, salidas de API y servicios instalables de IBM MQ

Este tema contiene enlaces a información sobre el uso y desarrollo de estos programas.

Para obtener información sobre cómo puede utilizar salidas de usuario, salidas de API y servicios instalables para ampliar los servicios del gestor de colas, consulte Ampliación de los servicios del gestor de colas.

Para obtener información sobre la escritura y compilación de salidas y servicios instalables, consulte los subtemas.

Conceptos relacionados
Programas de salida de canal para canales MQI
Referencia relacionada
Referencia a la salida de la API
Información de consulta sobre la interfaz de servicios instalables
IBM i Información de consulta sobre la interfaz de servicios instalables en IBM i

ALW Escritura de salidas y servicios instalables en AIX, Linux, and Windows

Puede escribir y compilar salidas sin enlazar con ninguna biblioteca de IBM MQ en AIX, Linux, and Windows.

Acerca de esta tarea

Este tema se aplica solo a sistemas AIX, Linux, and Windows. Para obtener información detallada sobre cómo escribir salidas y servicios instalables en otras plataformas, consulte los temas específicos de la correspondiente plataforma.

Si IBM MQ se ha instalado en una ubicación no predeterminada, hay que escribir y compilar las salidas sin enlazarlas con ninguna biblioteca de IBM MQ.

Se pueden escribir y compilar salidas en sistemas AIX, Linux, and Windows sin enlazar con ninguna de estas bibliotecas de IBM MQ:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Las salidas existentes que están enlazadas a estas bibliotecas siguen funcionando, siempre y cuando en los sistemas AIX and Linux esté instalado IBM MQ en la ubicación predeterminada.

Procedimiento

1. Incluya el archivo de cabecera cmqec.h.

   La inclusión de este archivo de cabecera incluye automáticamente los archivos de cabecera cmqc.h, cmqxc.h y cmqzc.h.
2. Escriba la salida de forma que las llamadas MQI y DCI se realicen a través de la estructura MQIEP. Para obtener más información sobre la estructura MQIEP, consulte Estructura MQIEP.
   - Servicios instalables
     - Utilice el parámetro **Hconfig** para que apunte a la llamada MQZEP.
     - Hay que comprobar que los primeros 4 bytes de **Hconfig** coinciden con el **StrucId** de la estructura MQIEP antes de utilizar el parámetro **Hconfig**.
     - Para obtener más información sobre cómo desarrollar componentes de servicio instalable, consulte MQIEP.
   - Salidas de API
     - Utilice el parámetro **Hconfig** para que apunte a la llamada MQXEP.
     - Hay que comprobar que los primeros 4 bytes de **Hconfig** coinciden con el **StrucId** de la estructura MQIEP antes de utilizar el parámetro **Hconfig**.
     - Para obtener más información sobre cómo desarrollar salidas de API, consulte “Desarrollo de una salida de API” en la página 979.
   - Salidas de canal
     - Utilice el parámetro **pEntryPoints** de la estructura MQCXP para que apunte a las llamadas MQI y DCI.
     - Hay que comprobar que el número de versión de MQCXP sea como mínimo 8 o superior antes de utilizar **pEntryPoints**.
     - Para obtener más información sobre cómo desarrollar salidas de canal, consulte “Cómo escribir programas de salida de canal” en la página 989.
   - Salidas de conversión de datos
     - Utilice el parámetro **pEntryPoints** de la estructura MQDXP para que apunte a las llamadas MQI y DCI.

- Hay que comprobar que el número de versión de MQDXP sea como mínimo 2 o superior antes de utilizar **pEntryPoints**.
- Puede utilizar el comando **crtmqcvx** y el archivo de fuente amqsvfc0.c para crear un código de conversión de datos que utilice el parámetro **pEntryPoints**. Consulte los apartados "Desarrollo de una salida de conversión de datos para IBM MQ for Windows" en la página 1016 y "Escritura de una salida de conversión de datos para sistemas IBM MQ for AIX or Linux" en la página 1013.
- Si tiene salidas de conversión de datos existentes que se generaron con el comando **crtmqcvx**, hay que regenerar la salida con el comando actualizado.
- Para obtener más información sobre cómo escribir salidas de conversión de datos, consulte "Escribir salidas de conversión de datos" en la página 1009.

- Salidas previas a la conexión
  - Utilice el parámetro **pEntryPoints** de la estructura MQNXP para que apunte a las llamadas MQI y DCI.
  - Hay que comprobar que el número de versión de MQNXP sea como mínimo 2 o superior antes de utilizar **pEntryPoints**.
  - Para obtener más información sobre cómo desarrollar salidas previas a la conexión, consulte "Referencia a las definiciones de conexión utilizando una salida de preconexión desde un depósito" en la página 1018.
- Salidas de publicación
  - Utilice el parámetro **pEntryPoints** de la estructura MQPSXP para que apunte a las llamadas MQI y DCI.
  - Hay que comprobar que el número de versión de MQPSXP sea como mínimo 2 o superior antes de utilizar **pEntryPoints**.
  - Para obtener más información sobre cómo desarrollar salidas de publicación, consulte "Desarrollo y compilación de una salida de publicación" en la página 1020.
- Salidas de carga de trabajo de clúster
  - Utilice el parámetro **pEntryPoints** de la estructura MQWXP para que apunte a llamadas MQXCLWLN.
  - Hay que comprobar que el número de versión de MQWXP sea como mínimo 4 o superior antes de utilizar **pEntryPoints**.
  - Para obtener más información sobre cómo escribir salidas de carga de trabajo de clúster, consulte "Escritura y compilación de salidas de carga de trabajo de clúster" en la página 1022.

Por ejemplo, en una salida de canal que llame a MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Podrá encontrar más ejemplos en "Utilización de programas procedimentales de ejemplo de IBM MQ" en la página 1085.

3. Compile la salida:
   - No enlace con las bibliotecas de IBM MQ.
   - No incluya una RPath incorporada en ninguna biblioteca de IBM MQ en su salida.
   - Para obtener más información sobre la compilación de la salida, consulte uno de los temas siguientes:
     - Salidas de API: "Compilación de salidas de API" en la página 980.

- Salidas de canal, salidas de publicación, salidas de carga de trabajo de clúster: “Compilación de programas de salida de canal en sistemas AIX, Linux, and Windows” en la página 1007.
- Salidas de conversión de datos: “Escribir salidas de conversión de datos” en la página 1009.

4. Coloque la salida en uno de los siguientes lugares:
   - Una ruta de su elección que esté plenamente cualificada al configurar la salida
   - La ruta de salida predeterminada, en un directorio de instalación específico. Por ejemplo, *MQ_DATA_PATH*/exits/installation2.
   - La ruta de salida predeterminada

     La ruta salida predeterminada es *MQ_DATA_PATH*/exits para salidas de 32 y *MQ_DATA_PATH*/exits64 para salidas de 64 bits. Puede cambiar estas rutas en los archivos qm.ini o mqclient.ini. Para obtener más información, consulte Ruta de salida. En Windows y Linux, puede utilizar IBM MQ Explorer para cambiar la vía de acceso:

     a. Pulse con el botón derecho en el nombre del gestor de colas
     b. Pulse **Propiedades...**
     c. Pulse **Salidas**
     d. En el campo de ruta predeterminada de las salidas, especifique el nombre de ruta del directorio que contiene el programa de salida.

   Si una salida se coloca en un directorio de instalación específico y también en el directorio de vía de acceso predeterminado, la salida del directorio de instalación especifica es utilizada por la instalación del IBM MQ especificado en la vía de acceso. Por ejemplo, la salida se coloca en /exits/installation2 y en /exits, pero no en /exits/installation1. La instalación de IBM MQ installation2 utiliza la salida de /exits/installation2. La instalación de IBM MQ installation1 utiliza la salida del directorio /exits.
5. Si es necesario, configure la salida:
   - Servicio instalables: “Configuración de servicios y componentes” en la página 971.
   - Salidas de API: “Configurar salidas de API” en la página 983.
   - Salidas de canal: “Configuración de una salida de canal” en la página 1008.
   - Salidas de publicación: “Configuración de una salida de publicación” en la página 1021.
   - Salidas previas a la conexión: Stanza PreConnect del archivo de configuración de cliente.

ALW *Salidas de API no enlazadas con una biblioteca de MQI*

Bajo determinadas circunstancias, se debe enlazar la salida de la API existente, que no se puede volver a codificar para utilizar los punteros de función MQIEP, con una biblioteca de la API de IBM MQ.

Esto es necesario para que el enlazador de entorno de ejecución del sistema pueda cargar la salida de API existente en programas que aún no tienen los punteros de función cargados.

Nota: Esta información se limita a las salidas de API existentes que hacen llamadas MQI directamente. Es decir, las salidas que no utilizan MQIEP. En la medida de lo posible, hay que contar con que haya que volver a codificar la salida para que use los puntos de entrada MQIEP en su lugar.

Desde IBM MQ 8.0, **runmqsc** es un ejemplo de un programa que no se enlaza directamente con una biblioteca MQI.

Por lo tanto, una salida de API que no se ha enlazado con su biblioteca de API IBM MQ necesaria, o que se ha vuelto a codificar para utilizar MQIEP, no se puede cargar en **runmqsc**.

Verá errores en el registro de errores del gestor de colas, por ejemplo, AMQ6175: `El sistema no ha podido cargar dinámicamente la biblioteca compartida`, junto con el texto calificado como, por ejemplo, `undefined symbol: MQCONN`.

y AMQ7214: `El módulo de la salida de API 'nombresalida' no se ha podido cargar`.

Tareas relacionadas

"Escritura de salidas y servicios instalables en AIX, Linux, and Windows" en la página 960
Puede escribir y compilar salidas sin enlazar con ninguna biblioteca de IBM MQ en AIX, Linux, and Windows.

ALW Servicios y componentes instalables para AIX, Linux, and Windows

En esta sección, se describen los servicios instalables y las funciones y componentes asociados con ellos. La interfaz de estas funciones se describe para que usted o los proveedores del software puedan proporcionar los componentes.

Los principales motivos para proporcionar los servicios instalables de IBM MQ son:

- Tener flexibilidad para elegir si desea utilizar los componentes proporcionados por los productos de IBM MQ, o bien sustituirlos o aumentarlos con otros.
- Permitir que los proveedores participen, proporcionando componentes que pueden utilizar nuevas tecnologías, sin realizar cambios internos en los productos de IBM MQ.
- Permitir que IBM MQ utilice las nuevas tecnologías de forma más rápida y barata, para proporcionar productos antes a precios más bajos.

Los *servicios instalables* y los *componentes de servicio* forman parte de la estructura de productos de IBM MQ. En el centro de esta estructura está la parte del gestor de colas que implementa las funciones y las reglas asociadas con la Interfaz de colas de mensajes (MQI). Esta parte central requiere una serie de funciones de servicios, denominadas *servicios instalables*, para poder realizar su trabajo. Los servicios instalables son:

- Servicio de autorización
- Servicio de nombres

Cada servicio instalable es un conjunto relacionado de funciones que se implementan utilizando uno o varios *componentes de servicio*. Cada componente se invoca utilizando una interfaz debidamente diseñada y de disponibilidad pública. Esto permite a los proveedores de software independientes y a otros proveedores de terceros proporcionar componentes instalables para aumentar o sustituir los suministrados por los productos de IBM MQ. En la Tabla 138 en la página 964, se resumen los servicios y componentes que pueden utilizarse.

Tabla 138. Resumen de los componentes de servicio instalables

Servicio instalable	Componente suministrado	Función	Requisitos
Servicio de autorización	gestor de autorizaciones sobre objetos (OAM)	Proporciona la comprobación de autorización en mandatos y llamadas MQI. Los usuarios pueden escribir sus propios componentes para aumentar o sustituir el OAM. Por ejemplo, para comprobar que un ID de usuario tiene autorización para abrir una cola.	(Se supone que existen los recursos de autorización apropiados para la plataforma)
Servicio de nombres	Ninguna	Proporciona soporte para que el gestor de colas pueda buscar el nombre del gestor de colas que es el propietario de una cola especificada. • Definida por el usuario	• Un gestor de nombres de terceros o escrito por el usuario

La interfaz de servicios instalables se describe en Información de consulta sobre la interfaz de servicios instalables.

Tareas relacionadas

Configuración de servicios instalables

Desarrollo de un componente de servicio

En esta sección se describe la relación entre servicios, componentes, puntos de entrada y códigos de retorno.

Funciones y componentes

Cada servicio consta de un conjunto de funciones relacionadas. Por ejemplo, el servicio de nombres incluye funciones para:

- Buscar un nombre de cola y devolver el nombre del gestor de colas en el que está definida la cola.
- Insertar un nombre de cola en el directorio del servicio.
- Borrar un nombre de cola del directorio del servicio.

También contiene funciones de inicialización y terminación.

Un servicio instalable se proporciona mediante uno o más componentes de servicio. Cada componente puede realizar algunas o todas las funciones que se han definido para dicho servicio. Por ejemplo, en IBM MQ for AIX, el componente de servicio de autorización proporcionado, el OAM, realiza todas las funciones disponibles. Consulte “Interfaz del servicio de autorización” en la página 968 para obtener más información. El componente también es responsable de la gestión de los recursos o software subyacentes (por ejemplo, un directorio LDAP) que necesite para implementar el servicio. Los archivos de configuración proporcionan un modo estándar para cargar el componente y determinar las direcciones de las rutinas de función que proporciona.

En Figura 99 en la página 965 se muestra cómo se relacionan servicios y componentes:

- Un servicio se define en un gestor de colas mediante las stanzas de un archivo de configuración.
- Cada servicio está soportado por código suministrado en el gestor de colas. Los usuarios no pueden modificar este código y, por tanto, no pueden crear sus propios servicios.
- Cada servicio se implementa mediante uno o varios componentes. Estos se pueden suministrar con el producto o pueden estar escritos por el usuario. Se pueden invocar varios componentes de un servicio, soportando cada uno de ellos diferentes recursos dentro del servicio.
- Los puntos de entrada conectan los componentes de servicio con el código de soporte del gestor de colas.

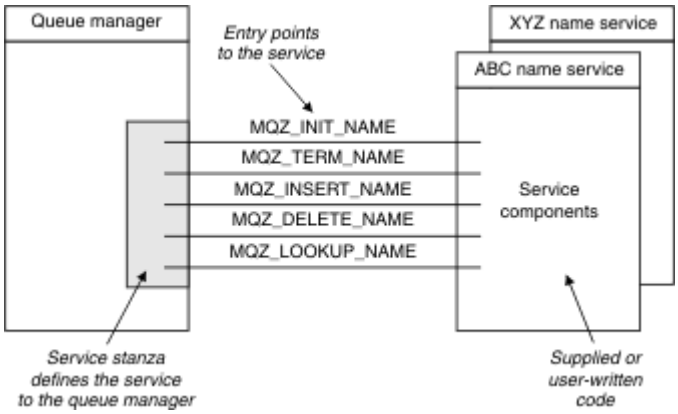


Figura 99. Servicios, componentes y puntos de entrada

Puntos de entrada

Cada componente de servicio se representa mediante una lista de las direcciones de punto de entrada de las rutinas que soportan un servicio instalable concreto. El servicio instalable define la función que tiene que realizar cada rutina.

El orden en el que se configuran los componentes de servicio define el orden en que se invocan los puntos de entrada cuando se intenta atender una petición del servicio.

En el archivo de cabecera que se proporciona, `cmqzc.h`, los puntos de entrada proporcionados para cada servicio tienen un prefijo MQZID_.

Si los servicios están presentes, se cargarán en un orden predefinido. La lista siguiente muestra los servicios y el orden en el que se inicializan.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` es el único servicio que está configurado de forma predeterminada. Configure manualmente `NameService` y `UserIdentifierService` si desea utilizarlos.

Los servicios y los componentes de servicio tienen una correlación de uno a uno o de uno a varios. Se pueden definir varios componentes de servicio para cada servicio. En los sistemas AIX and Linux, el valor de servicio de la stanza `ServiceComponent` debe coincidir con el valor de nombre de la stanza de servicio en el archivo qm.ini. En Windows, el valor de clave de registro de servicio de `ServiceComponent` debe coincidir con el valor de clave de registro de nombre y se define como: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` donde *nombreqm* es el nombre del gestor de colas.

En los sistemas AIX and Linux, los componentes de servicio se inician en el orden en que están definidos en el archivo qm.ini. En Windows, puesto que se utiliza el registro Windows, IBM MQ emite una llamada **RegEnumKey** que devuelve los valores en orden alfabético. Por lo tanto, en Windows, los servicios se llaman en orden alfabético, tal como están definidos en el registro.

El orden de las definiciones de `ServiceComponent` es significativo. Este orden determina el orden en el que se ejecutan los componentes de un servicio determinado. Por ejemplo, `AuthorizationService` en Windows se configura con el componente OAM predeterminado llamado `MQSeries.WindowsNT.auth.service`. Se pueden definir componentes adicionales para sustituir el OAM predeterminado. A menos que se especifique `MQCACF_SERVICE_COMPONENT`, se utilizará el primer componente detectado por orden alfabético para procesar la solicitud y se utilizará el nombre de dicho componente.

Códigos de retorno

Los componentes de servicio proporcionan códigos de retorno al gestor de colas para informar de diversas condiciones. Informan sobre el éxito o fracaso de la operación, e indican si el gestor de colas tiene que proseguir con el siguiente componente de servicio. Esta indicación se incluye en un parámetro aparte, *Continuation*.

Datos de un componente

Es posible que un único servicio necesite compartir datos entre sus diferentes funciones. Los servicios instalables proporcionan un área de datos opcional que se pasa en cada invocación de un componente de servicio. Esta área de datos es de uso exclusivo del componente de servicio. La comparten todas las invocaciones de una función específica, incluso si se efectúan desde espacios de direcciones o procesos diferentes. Se garantiza que es direccionable desde el componente de servicio siempre que se invoque. Debe declarar el tamaño de esta área en la stanza *`ServiceComponent`* .

Inicialización y terminación de componentes
Uso de las opciones de inicialización y terminación de componentes.

Cuando se invoca la rutina de inicialización de componente, hay que invocar la función **MQZEP** del gestor de colas por cada punto de entrada soportado por el componente. **MQZEP** define un punto de entrada al servicio. Se presupone que todos los puntos de salida no definidos son NULL.

Un componente siempre se invoca una vez con la opción de inicialización primaria antes de ser invocado de otra manera.

En determinadas plataformas, se puede invocar un componente con la opción de inicialización secundaria. Por ejemplo, puede invocarse una vez por cada tarea, hebra o proceso de sistema operativo a través de los cuales se accede al servicio.

Si se utiliza la inicialización secundaria:

- Se puede invocar el componente más de una vez en la inicialización secundaria. Por cada llamada de este tipo, se emite una llamada coincidente para la terminación secundaria cuando el servicio ya no es necesario.

  En los servicios de nombres, esta llamada es MQZ_TERM_NAME.

  En los servicios de autorización, esta llamada es MQZ_TERM_AUTHORITY.
- Cada vez que se llama al componente para las inicializaciones primaria y secundaria, hay que volver a especificar los puntos de entrada (llamando a MQZEP).
- Solo se utiliza una copia de los datos del componente; no hay una copia distinta para cada inicialización secundaria.
- El componente no se invoca para ninguna otra llamada al servicio (desde el proceso del sistema operativo, hebra o tarea, según corresponda) antes que se lleve a cabo la inicialización secundaria.
- El componente tiene que establecer el parámetro **Version** al mismo valor en las inicializaciones primaria y secundaria.

El componente siempre se invoca con la opción de terminación primaria una vez, cuando ya no es necesario. No se realizan más llamadas a este componente.

El componente se invoca con la opción de terminación secundaria si se ha invocado para la inicialización secundaria.

Gestor de autorizaciones sobre objetos (OAM)
El componente de servicio de autorización que se proporciona con los productos de IBM MQ se denomina Gestor de autorizaciones sobre objetos (OAM).

De forma predeterminada, OAM está activo y funciona con los comando de control **dspmqaut** (mostrar autorización), **dmpmqaut** (volcar autorización) y **setmqaut** (establecer o restablecer autorización).

La sintaxis de estos mandatos y cómo utilizarlos se describen en Administración de IBM MQ for Multiplatforms utilizando mandatos de control.

El OAM funciona con la *entidad* de un principal o grupo:

- Linux AIX En sistemas AIX and Linux, el principal es un ID de usuario o un ID asociado a un programa de aplicación que se ejecuta en nombre de un usuario. Un grupo es una recopilación de principales definida por el sistema.
- Windows En sistemas Windows, el principal es un ID de usuario de Windows o un ID asociado a un programa de aplicación que se ejecuta en nombre de un usuario. Un grupo es un grupo de Windows.

Las autorizaciones se pueden otorgar o revocar a nivel de principal o de grupo.

Cuando se realiza una petición MQI o se emite un comando, el OAM comprueba la entidad asociada a la operación que tiene autorización para realizar la operación solicitada y para acceder a los recursos del gestor de colas especificados.

El servicio de autorizaciones permite aumentar o sustituir la comprobación de la autoridad que proporcionan los gestores de colas escribiendo su propio componente de servicio de autorizaciones.

Servicio de nombres
El servicio de nombres es un servicio instalable que proporciona soporte al gestor de colas para buscar el nombre del gestor de colas que es propietario de una cola especificada. No se puede recuperar ningún otro atributo de cola de un servicio de nombres.

El servicio de nombres permite a una aplicación abrir colas remotas para la salida como si fueran colas locales. Un servicio de nombres no se invoca para objetos distintos de colas.

Nota: Las colas remotas han de tener el atributo **Scope** establecido a CELL.

Cuando una aplicación abre una cola, primero busca su nombre en el directorio del gestor de colas. Si no la encuentra allí, busca en tantos servicios de nombres como se hayan configurado hasta encontrar uno que reconozca el nombre de la cola. Si ninguno reconoce el nombre, la apertura falla.

El servicio de nombres devuelve el gestor de colas propietario de dicha cola. Luego, el gestor de colas continúa con la petición MQOPEN como si el comando hubiera especificado el nombre de la cola y del gestor de colas en la petición original.

Interfaz de servicio de nombres (Name Service Interface, NIS) del entorno IBM MQ.

Cómo funciona el servicio de nombres

Si una definición de cola especifica el atributo **Scope** como gestor de colas, es decir, SCOPE(QMGR) en MQSC, la definición de cola (junto con todos los atributos de cola) solo se almacenan en el directorio del gestor de colas. Esto no se puede sustituir por un servicio instalable.

Si una definición de cola especifica el atributo **Scope** como célula, es decir, SCOPE(CELL) en MQSC, la definición de la cola se vuelve a almacenar en el directorio del gestor de colas, junto con todos los atributos de cola. Sin embargo, la cola y el nombre del gestor de colas también se almacenan en un servicio de nombres. Si no hay ningún servicio disponible que pueda almacenar esta información, no se puede definir una cola con la célula *Scope* .

El directorio en el que se almacena la información lo puede gestionar el servicio o bien este se puede apoyar en un servicio subyacente como, por ejemplo, un directorio LDAP, a tal fin. En cualquiera de los casos, las definiciones almacenadas en el directorio tienen tendrán que persistir, incluso después de que el componente y el gestor de colas hayan terminado, hasta que se borren explícitamente.

Nota:

1. Para enviar un mensaje a una definición de cola local de un host remoto (con un ámbito de CELL) en un gestor de colas distinto dentro de una célula de directorio de denominación, hay que definir un canal.
2. No se pueden obtener mensajes directamente de la cola remota, incluso si tiene un ámbito de CELL.
3. No se necesita ninguna definición de cola remota cuando se envía a una cola con un ámbito de CELL.
4. El servicio de denominación define centralmente la cola de destino, aunque todavía se necesitan una cola de transmisión para el gestor de colas de destino y un par de definiciones de canal. Ademaás la cola de transmisión del sistema local ha de tener el mismo nombre que el que tiene el gestor de colas propietario de la cola de destino, con ámbito de celda, en el sistema remoto.

   Por ejemplo, si el gestor de colas remoto tiene el nombre QM01, la cola de transmisión en el sistema local también habrá de tener el nombre QM01.

Interfaz del servicio de autorización
El servicio de autorización proporciona puntos de entrada para que lo utilice un gestor de colas.

Los puntos de entrada son los siguientes:

MQZ_AUTHENTICATE_USER
Autentica un ID de usuario y contraseña, y puede establecer la identidad de los campos de contexto.

MQZ_CHECK_AUTHORITY
Comprueba si una entidad tiene autorización para realizar una o varias operaciones en un objeto especificado.

MQZ_CHECK_PRIVILEGED
Comprueba si un usuario especificado es privilegiado.

MQZ_COPY_ALL_AUTHORITY
Copia todas las autorizaciones actuales que existen de un objeto referenciado a otro objeto.

MQZ_DELETE_AUTHORITY
Borra todas las autorizaciones asociadas a un objeto especificado.

MQZ_ENUMERATE_AUTHORITY_DATA
Recupera todos los datos de autorización que coinciden con los criterios de selección especificados.

MQZ_FREE_USER
Libera recursos asignados asociados.

MQZ_GET_AUTHORITY
Obtiene la autorización que una entidad tiene para acceder a un objeto especificado.

MQZ_GET_EXPLICIT_AUTHORITY
Obtiene la autorización que tiene un grupo nombrado para acceder a un objeto especificado (pero sin la autorización adicional **nobody** del grupo) o a la autorización que el grupo primario del principal nombrado tiene para acceder a un objeto especificado.

MQZ_INIT_AUTHORITY
Inicializa el componente de servicio de autorizaciones.

MQZ_INQUIRE
Consulta la funcionalidad soportada del servicio de autorizaciones.

MQZ_REFRESH_CACHE
Renueva todas las autorizaciones.

MQZ_SET_AUTHORITY
Establece la autorización que una entidad tiene sobre un objeto especificado.

MQZ_TERM_AUTHORITY
Finaliza el componente de servicio de autorizaciones.

Además, en IBM MQ for Windows, el servicio de autorización proporciona los puntos de entrada siguientes para que los utilice el gestor de colas:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Estos puntos de entrada soportan el uso del identificador de seguridad de Windows (NT SID).

Estos nombres se definen como **typedef** s, en el archivo de cabecera `cmqzc.h`, que se puede utilizar para crear un prototipo de las funciones de componente.

La función de inicialización (**MQZ_INIT_AUTHORITY**) debe ser el punto de entrada principal para el componente. Las demás funciones se invocan a través de la dirección de punto de entrada que la función de inicialización ha añadido en el vector del punto de entrada del componente.

Interfaz de servicio de nombres
Un servicio de nombres proporciona puntos de entrada para que los use el gestor de colas.

Se proporcionan los puntos de entrada siguientes:

MQZ_INIT_NAME
Inicializa el componente de servicio de nombres.

MQZ_TERM_NAME
Finaliza el componente de servicio de nombres.

MQZ_LOOKUP_NAME
Busca el nombre de gestor de colas de la cola especificada.

MQZ_INSERT_NAME
Inserta una entrada que contiene el nombre del gestor de colas propietario para la cola especificada en el directorio utilizado por el servicio.

MQZ_DELETE_NAME
Borra la entrada de la cola especificada del directorio utilizado por el servicio.

Si hay más de un servicio de nombres configurado:

- En búsquedas, se invoca la función MQZ_LOOKUP_NAME por cada servicio de la lista hasta que se resuelve el nombre de la cola (a menos que algún componente indique que hay que parar la búsqueda).
- En inserciones, se invoca la función MQZ_INSERT_NAME para el primer servicio de la lista que soporte esta función.
- En borrados, se invoca la función MQZ_DELETE_NAME para el primer servicio de la lista que soporte esta función.

No tenga más de un componente que soporte las funciones de inserción y borrado. No obstante, un componente que solamente soporte búsquedas es factible y puede utilizarse, por ejemplo, como último componente de la lista para resolver cualquier nombre que otro componente del servicio de nombres no conozca a un gestor de colas en el que puede definirse el nombre.

En C, los nombres se definen como tipos de datos de función utilizando la sentencia typedef. Se pueden utilizar para crear un prototipo de las funciones de servicio a fin de garantizar que los parámetros sean correctos.

El archivo de cabecera que contiene todo el material específico de los servicios instalables es `cmqzc.h` para C.

Aparte de la función de inicialización (MQZ_INIT_NAME), que tiene que ser el punto de entrada principal del componente, las funciones se invocan mediante la dirección de punto de entrada añadida por la función de inicialización, utilizando la llamada MQZEP.

Utilización de varios componentes de servicio
Puede instalar más de un componente para un servicio. De este modo, los componentes pueden proporcionar solamente implementaciones parciales del servicio y confiar en otros componentes para que proporcionen las funciones restantes.

Ejemplo de cómo utilizar varios componentes

Supongamos que crea dos componentes de servicios de nombres denominados `ABC_name_serv` y `XYZ_name_serv`.

`ABC_name_serv`
Este componente dará soporte la inserción de un nombre en el directorio del servicio, o la supresión del nombre del mismo, pero no soporta la búsqueda un nombre de cola.

`XYZ_name_serv`
Este componente dará soporte a la búsqueda de un nombre de cola pero no dará soporte a la inserción de un nombre en el directorio del servicio ni la supresión de un nombre del mismo.

El componente `ABC_name_serv` contiene una base de datos de nombres de cola y utiliza dos algoritmos simples para insertar o suprimir un nombre del directorio de servicio.

El componente `XYZ_name_serv` utiliza un algoritmo simple que devuelve un nombre de gestor de colas fijo para cualquier nombre de cola con el que se invoque. No mantiene una base de datos de nombres de colas y, por lo tanto, no da soporte a las funciones de inserción y supresión.

Los componentes están instalados en el mismo gestor de colas. Las stanzas *`ServiceComponent`* se ordenan para que el componente `ABC_name_serv` se invoque primero. Las llamadas para insertar o suprimir una cola en un directorio de componentes las maneja el componente `ABC_name_serv` ; es el único que implementa estas funciones. Sin embargo, una llamada de búsqueda que el componente

ABC_name_serv no puede resolver se pasa al componente de sólo búsqueda, XYZ_name_serv. Este componente proporciona un nombre de gestor de colas a partir de su algoritmo simple.

Omisión de puntos de entrada cuando se utilizan varios componentes

Si decide utilizar varios componentes para proporcionar un servicio, puede diseñar un componente de servicio que no implemente determinadas funciones. La infraestructura de servicios instalables no impone ninguna limitación en cuanto a lo que puede omitir. Sin embargo, para los servicios instalables específicos, omitir una o varias funciones puede generar una incoherencia lógica en cuanto a la finalidad del servicio.

Ejemplo de puntos de entrada con varios componentes

La Tabla 139 en la página 971 muestra un ejemplo de un servicio de nombres instalable para el que se han instalado los dos componentes. Cada uno de ellos da soporte a un conjunto de funciones diferentes asociado a este servicio instalable en concreto. Para la función de insertar, se invoca en primer lugar el punto de entrada del componente ABC. Se presupone que los puntos de entrada que no se han definido en el servicio (utilizando **MQZEP**) son NULL. En la tabla se proporciona un punto de entrada para la inicialización, pero esto no es necesario porque la inicialización la lleva a cabo el punto de entrada principal del componente.

Cuando el gestor de colas tenga que utilizar un servicio instalable, utilizará los puntos de entrada definidos para dicho servicio (las columnas de Tabla 139 en la página 971). El gestor de colas toma cada uno de los componentes según el orden en que aparecen y determina la dirección de la rutina que implementa la función necesaria. A continuación, llama a la rutina si ésta existe. Si la operación se ejecuta correctamente, el gestor de colas utilizará cualquier resultado e información de estado.

Tabla 139. Ejemplo de puntos de entrada de un servicio instalable

Número de función	**Componente de servicio de nombres ABC**	**Componente de servicio de nombres XYZ**
MQZID_INIT_NAME (Inicializar)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Finalizar)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Insertar)	ABC_Insert()	Nulo
MQZID_DELETE_NAME (Suprimir)	ABC_Delete()	Nulo
MQZID_LOOKUP_NAME (Buscar)	Nulo	XYZ_Lookup()

Si la rutina no existe, el gestor de colas repite este proceso para el componente siguiente de la lista. Asimismo, si la rutina existe pero devuelve un código indicando que no puede realizar la operación, el intento continúa con el siguiente componente que hay disponible. La rutina de los componentes de servicio puede devolver un código que indique que no deben realizarse intentos adicionales de realizar la operación.

Configuración de servicios y componentes

Puede configurar los componentes de servicio utilizando los archivos de configuración del gestor de colas, excepto en los sistemas Windows, donde cada gestor de colas tiene su propia stanza en el registro.

Procedimiento

1. Añada stanzas al archivo de configuración del gestor de colas, qm.ini, para definir el servicio en el gestor de colas y especificar la ubicación del módulo:
   - Todo servicio utilizado habrá de tener una sección Service que defina el servicio al gestor de colas. Para obtener más información, consulte Stanza de servicio del archivo qm.ini.

- Por cada componente de un servicio, tiene que haber una stanza `ServiceComponent`. Esta stanza identifica el nombre y la vía de acceso del módulo que contiene el código para ese componente. Para obtener más información, consulte la sección ServiceComponent del archivo qm.ini.

  El componente de servicio de autorización, conocido como gestor de autorizaciones sobre objetos (OAM), se proporciona con el producto. Cuando se crea un gestor de colas, el archivo de configuración del gestor de colas (o el Registro en sistemas Windows) se actualiza automáticamente para incluir las stanzas apropiadas para el servicio de autorización y para el componente predeterminado (el OAM). Para los demás componentes, hay que configurar manualmente el archivo de configuración del gestor de colas.

  El código de cada componente de servicio se carga en el gestor de colas al iniciarse este, utilizando enlaces dinámicos siempre que la plataforma lo soporte.

2. Pare y reinicie el gestor de colas para activar el componente.

Referencia relacionada

Stanza de servicio del archivo qm.ini

Stanza ServiceComponent del archivo qm.ini

Renovación del gestor de autorizaciones sobre objetos (OAM) después de cambiar la autorización de un usuario

En IBM MQ, puede renovar la información del grupo de autorización del OAM inmediatamente después de cambiar la pertenencia al grupo de autorización de un usuario, para reflejar los cambios realizados a nivel de sistema operativo sin necesidad de detener y reiniciar el gestor de colas. Para ello, emita el mandato **REFRESH SECURITY**.

Nota: Cuando cambia las autorizaciones con el mandato `setmqaut`, el OAM implementa dichos cambios inmediatamente.

Los gestores de colas almacenan los datos de autorización en una cola local denominada SYSTEM.AUTH.DATA.QUEUE. Estos datos se gestionan mediante **amqzfuma.exe**.

Referencia relacionada

REFRESH SECURITY

IBM i Servicios instalables y componentes en IBM i

Aquí puede obtener más información acerca de los servicios instalables y las funciones y componentes asociados a los mismos. La interfaz de estas funciones se describe para que usted o los proveedores del software puedan proporcionar los componentes.

Los principales motivos para proporcionar los servicios instalables de IBM MQ son:

- Para proporcionarle la flexibilidad de elegir si desea utilizar componentes proporcionados por IBM MQ for IBM i, o bien sustituir o aumentarlos con otros.
- Para permitir que los proveedores participen, proporcionando componentes que pueden utilizar nuevas tecnologías, sin realizar cambios internos en IBM MQ for IBM i.
- Permitir que IBM MQ utilice las nuevas tecnologías de forma más rápida y barata, para proporcionar productos antes a precios más bajos.

Los *servicios instalables* y los *componentes de servicio* forman parte de la estructura de productos de IBM MQ. En el centro de esta estructura está la parte del gestor de colas que implementa las funciones y las reglas asociadas con la Interfaz de colas de mensajes (MQI). Esta parte central requiere una serie de funciones de servicios, denominadas *servicios instalables*, para poder realizar su trabajo. El servicio instalable disponible en IBM MQ for IBM i es el servicio de autorización.

Cada servicio instalable es un conjunto relacionado de funciones que se implementan utilizando uno o varios *componentes de servicio*. Cada componente se invoca utilizando una interfaz debidamente diseñada y de disponibilidad pública. Esto permite a los distribuidores de software independiente y a terceros proporcionar componentes instalables para aumentar o sustituir los proporcionados por IBM MQ for IBM i. Tabla 140 en la página 973 resume el soporte para el servicio de autorización.

Tabla 140. Resumen de componentes del servicio de autorización		
Componente suministrado	**Función**	**Requisitos**
Gestor de autorizaciones sobre objetos (OAM)	Proporciona la comprobación de autorización en mandatos y llamadas MQI. Los usuarios pueden escribir sus propios componentes para aumentar o sustituir el OAM.	(Se supone que existen los recursos de autorización apropiados para la plataforma)
Componente de servicio de nombres DCE **Nota:** DCE sólo se admite en las versiones de IBM MQ anteriores a la v6.0.	• Permite que los gestores de colas compartan colas o • Definida por el usuario **Nota:** Las colas compartidas deben tener el atributo **Scope** establecido en CELL.	• Se precisa DCE para el componente suministrado o bien • Un gestor de nombres de terceros o escrito por el usuario

IBM i *Funciones y componentes en IBM i*

Utilice esta información para comprender las funciones y los componentes, puntos de entrada, códigos de retorno y datos de componente que puede utilizar en IBM MQ for IBM i.

Cada servicio consta de un conjunto de funciones relacionadas. Por ejemplo, el servicio de nombres incluye funciones para:

- Buscar un nombre de cola y devolver el nombre del gestor de colas en el que está definida la cola.
- Insertar un nombre de cola en el directorio del servicio.
- Borrar un nombre de cola del directorio del servicio.

También contiene funciones de inicialización y terminación.

Un servicio instalable se proporciona mediante uno o más componentes de servicio. Cada componente puede realizar algunas o todas las funciones que se han definido para dicho servicio. El componente también es responsable de la gestión de los recursos o software subyacentes que necesite para implementar el servicio. Los archivos de configuración proporcionan un modo estándar para cargar el componente y determinar las direcciones de las rutinas de función que proporciona.

Los servicios y componentes están relacionados del modo siguiente:

- Un servicio se define en un gestor de colas mediante las stanzas de un archivo de configuración.
- Cada servicio está soportado por código suministrado en el gestor de colas. Los usuarios no pueden modificar este código y, por tanto, no pueden crear sus propios servicios.
- Cada servicio se implementa mediante uno o varios componentes. Estos se pueden suministrar con el producto o pueden estar escritos por el usuario. Se pueden invocar varios componentes de un servicio, soportando cada uno de ellos diferentes recursos dentro del servicio.
- Los puntos de entrada conectan los componentes de servicio con el código de soporte del gestor de colas.

Puntos de entrada

Cada componente de servicio se representa mediante una lista de las direcciones de punto de entrada de las rutinas que soportan un servicio instalable concreto. El servicio instalable define la función que tiene que realizar cada rutina. El orden en el que se configuran los componentes de servicio define el orden en que se invocan los puntos de entrada cuando se intenta atender una petición del servicio. En el archivo de cabecera que se proporciona, `cmqzc.h`, los puntos de entrada proporcionados para cada servicio tienen un prefijo MQZID_.

Códigos de retorno

Los componentes de servicio proporcionan códigos de retorno con el gestor de colas que informan sobre diversas situaciones. Informan sobre el éxito o fracaso de la operación, e indican si el gestor de colas tiene que proseguir con el siguiente componente de servicio. Esta indicación se incluye en un parámetro aparte, *Continuation*.

Datos de un componente

Es posible que un único servicio necesite compartir datos entre sus diferentes funciones. Los servicios instalables proporcionan un área de datos opcional que se pasa en cada invocación de un determinado componente de servicio. Esta área de datos es de uso exclusivo del componente de servicio. Es compartida por todas las invocaciones de una función determinada, incluso si se realizan desde distintos espacios de direcciones o procesos. Se garantiza que es direccionable desde el componente de servicio siempre que se invoque. Debe declarar el tamaño de esta área en la stanza *ServiceComponent* .

IBM i *Inicialización en IBM i*

Cuando se invoca la rutina de inicialización de componente, hay que invocar la función MQZEP del gestor de colas por cada punto de entrada soportado por el componente. MQZEP define un punto de entrada al servicio. Se presupone que todos los puntos de salida no definidos son NULL.

Inicialización primaria
Un componente siempre se invoca una vez con esta opción, antes de invocarse de cualquier otra forma.

Inicialización secundaria
Un componente puede invocarse con esta opción en determinadas plataformas. Por ejemplo, puede invocarse una vez por cada tarea, hebra o proceso de sistema operativo a través de los cuales se accede al servicio.

Si se utiliza la inicialización secundaria:

- Se puede invocar el componente más de una vez en la inicialización secundaria. Por cada llamada de este tipo, se emite una llamada coincidente para la terminación secundaria cuando el servicio ya no es necesario.

  En los servicios de autorización, esta llamada es MQZ_TERM_AUTHORITY.
- Cada vez que se llama al componente para las inicializaciones primaria y secundaria, hay que volver a especificar los puntos de entrada (llamando a MQZEP).
- Solo se utiliza una copia de los datos del componente; no hay una copia distinta para cada inicialización secundaria.
- El componente no se invoca para ninguna otra llamada al servicio (desde el proceso del sistema operativo, hebra o tarea, según corresponda) antes que se lleve a cabo la inicialización secundaria.
- El componente tiene que establecer el parámetro **Version** al mismo valor en las inicializaciones primaria y secundaria.

Terminación primaria
El componente siempre se inicia una vez con esta opción, cuando ya no se necesita más. No se realizan más llamadas a este componente.

Terminación secundaria
El componente se inicia con esta opción, si se ha iniciado para la inicialización secundaria.

IBM i *Configuración de servicios y componentes en IBM i*

Los componentes de servicio se configuran utilizando los archivos de configuración del gestor de colas.

Procedimiento

1. Añada stanzas al archivo de configuración del gestor de colas, `qm.ini`, para definir el servicio en el gestor de colas y especificar la ubicación del módulo:

- Todo servicio utilizado habrá de tener una sección `Service` que defina el servicio al gestor de colas. Para obtener más información, consulte Stanza de servicio del archivo qm.ini.
- Por cada componente de un servicio, tiene que haber una stanza `ServiceComponent`. Esta stanza identifica el nombre y la vía de acceso del módulo que contiene el código para ese componente. Para obtener más información, consulte la sección ServiceComponent del archivo qm.ini.

El componente de servicio de autorización, conocido como gestor de autorizaciones sobre objetos (OAM), se proporciona con el producto. Cuando crea un gestor de colas, el archivo de configuración del gestor de colas se actualiza automáticamente para incluir las stanzas correspondientes al servicio de autorización y al componente predeterminado (el OAM). Para los demás componentes, hay que configurar manualmente el archivo de configuración del gestor de colas.

El código de cada componente de servicio se carga en el gestor de colas al iniciarse este, utilizando enlaces dinámicos siempre que la plataforma lo soporte.

2.

IBM i *Creación de su propio componente de servicio en IBM i*

Use esta información para aprender a crear un componente de servicio en IBM MQ for IBM i.

Para crear su propio componente de servicio:

- Asegúrese de que el archivo de cabecera `cmqzc.h` está incluido en el programa.
- Cree la biblioteca compartida compilando el programa y enlazándolo con las bibliotecas compartidas `libmqm*` y `libmqmzf*`.

  Nota: Puesto que el agente puede ejecutarse en un entorno con hebras, hay que crear el OAM para que ejecute en un entorno con hebras. Esto incluye la utilización de las versiones con hebras de `libmqm` y `libmqmzf`.
- Añada stanzas al archivo de configuración del gestor de colas para definir el servicio al gestor de colas y para especificar la ubicación del módulo.
- Pare y reinicie el gestor de colas para activar el componente.

IBM i *Servicio de autorización en IBM i*

El servicio de autorización es un servicio instalable que permite a los gestores de colas invocar recursos de autorización, por ejemplo, comprobar que un ID de usuario tiene autorización para abrir una cola.

Este servicio es un componente de la interfaz de habilitación de seguridad (SEI) de IBM MQ, que forma parte de la infraestructura de IBM MQ. Se analizan los temas siguientes:

- "Gestor de autorizaciones sobre objetos (OAM)" en la página 975
- "Definición del servicio en el sistema operativo" en la página 976
- "Configuración de las stanzas del servicio de autorización" en la página 976
- "Interfaz de servicio de autorización en IBM i" en la página 976

Gestor de autorizaciones sobre objetos (OAM)

El componente de servicio de autorización que se proporciona con los productos de IBM MQ se denomina Gestor de autorizaciones sobre objetos (OAM). De forma predeterminada, el OAM está activo y funciona con los siguientes mandatos de control:

- **WRKMQMAUT**: trabajar con autorización
- **WRKMQMAUTD**: trabajar con datos de autorización
- **DSPMQMAUT**: visualizar autorización de objeto
- **GRTMQMAUT**: otorgar autorización de objeto
- **RVKMQMAUT**: revocar autorización de objeto
- **RFRMQMAUT**: renovar seguridad

Encontrará una descripción de la sintaxis de estos mandatos y cómo se utilizan en la ayuda de los mandatos CL. El OAM funciona con la *entidad* de un principal o grupo.

Cuando se realiza una solicitud MQI o se emite un mandato, el OAM comprueba la autorización de la entidad asociada con la operación para ver si puede realizar las siguientes acciones:

- Realizar la operación solicitada.
- Acceder a los recursos del gestor de colas especificados.

El servicio de autorizaciones permite aumentar o sustituir la comprobación de la autoridad que proporcionan los gestores de colas escribiendo su propio componente de servicio de autorizaciones.

Definición del servicio en el sistema operativo

Las stanzas del servicio de autorización contenidas en el archivo de configuración del gestor de colas `qm.ini` definen el servicio de autorización para el gestor de colas. Consulte "Configuración de servicios y componentes en IBM i" en la página 974 para obtener información sobre los tipos de stanza.

Configuración de las stanzas del servicio de autorización

En IBM MQ for IBM i:

Principal
Es un perfil de usuario del sistema IBM i.

Grupo
Es un perfil de grupo del sistema IBM i.

Las autorizaciones solo se pueden otorgar o revocar a nivel de grupo. Una solicitud para otorgar o revocar la autorización de un usuario actualiza el grupo primario de dicho usuario.

Cada gestor de colas tiene su propio archivo de configuración del gestor de colas. Por ejemplo, la vía de acceso y el nombre de archivo predeterminados del archivo de configuración del gestor de colas QMNAME es `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

La stanza *`Service`* y la stanza *`ServiceComponent`* para el componente de autorización predeterminado se añaden automáticamente a `qm.ini`, pero `WRKENVVAR` puede alterarlas temporalmente. Las demás stanzas *`ServiceComponent`* deben añadirse manualmente.

Por ejemplo, las siguientes stanzas en el archivo de configuración del gestor de colas definen dos componentes de servicio de autorización:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Figura 100. Stanzas del servicio de autorización en qm.ini en IBM i

La primera stanza del componente de servicio `MQ.UNIX.authorization.service` define el componente de servicio de autorización predeterminado, el OAM. Si elimina esta stanza y reinicia el gestor de colas, el OAM se inhabilita y no se realiza ninguna comprobación de autorización.

IBM i *Interfaz de servicio de autorización en IBM i*

La interfaz del servicio de autorización proporciona varios puntos de entrada para que los utilice el gestor de colas.

MQZ_AUTHENTICATE_USER
Autentica un ID de usuario y contraseña, y puede establecer la identidad de los campos de contexto.

MQZ_CHECK_AUTHORITY
Comprueba si una entidad tiene autorización para realizar una o varias operaciones en un objeto especificado.

MQZ_COPY_ALL_AUTHORITY
Copia todas las autorizaciones actuales que existen de un objeto referenciado a otro objeto.

MQZ_DELETE_AUTHORITY
Borra todas las autorizaciones asociadas a un objeto especificado.

MQZ_ENUMERATE_AUTHORITY_DATA
Recupera todos los datos de autorización que coinciden con los criterios de selección especificados.

MQZ_FREE_USER
Libera recursos asignados asociados.

MQZ_GET_AUTHORITY
Obtiene la autorización que una entidad tiene para acceder a un objeto especificado.

MQZ_GET_EXPLICIT_AUTHORITY
Obtiene la autorización que tiene un grupo nombrado para acceder a un objeto especificado (pero sin la autorización adicional **nobody** del grupo) o a la autorización que el grupo primario del principal nombrado tiene para acceder a un objeto especificado.

MQZ_INIT_AUTHORITY
Inicializa el componente de servicio de autorizaciones.

MQZ_INQUIRE
Consulta la funcionalidad soportada del servicio de autorizaciones.

MQZ_REFRESH_CACHE
Renueva todas las autorizaciones.

MQZ_SET_AUTHORITY
Establece la autorización que una entidad tiene sobre un objeto especificado.

MQZ_TERM_AUTHORITY
Finaliza el componente de servicio de autorizaciones.

Estos puntos de entrada soportan el uso del identificador de seguridad de Windows (NT SID).

Estos nombres se definen como **typedef** s, en el archivo de cabecera `cmqzc.h`, que se puede utilizar para crear un prototipo de las funciones de componente.

La función de inicialización (**MQZ_INIT_AUTHORITY**) debe ser el punto de entrada principal para el componente. Las demás funciones se invocan a través de la dirección de punto de entrada que la función de inicialización ha añadido en el vector del punto de entrada del componente.

Consulte “Creación de su propio componente de servicio en IBM i” en la página 975 para obtener más información.

Multi Escritura y compilación de salidas de API en Multiplatforms

Las salidas de API permiten escribir código que cambia el comportamiento de las llamadas de API de IBM MQ, como, por ejemplo, MQPUT y MQGET, e insertar ese código inmediatamente antes o inmediatamente después de esas llamadas.

Nota: z/OS No está soportado en IBM MQ for z/OS.

¿Por qué usar salidas de API?

Cada aplicación tiene un trabajo concreto que hacer y su código tiene que realizar dicha tarea de la forma más eficiente posible. En un nivel superior, es posible que desee aplicar procesos de empresa o procesos estándar a un gestor de colas determinado para todas las aplicaciones que utilicen este gestor de colas.

Resulta más eficaz hacerlo por encima del nivel de las aplicaciones individuales y, por tanto, sin tener que cambiar el código de cada aplicación afectada.

A continuación se ofrecen algunas sugerencias de áreas en las que las salidas de API podrían resultarle útiles:

Seguridad
En cuanto a la seguridad, puede proporcionar autenticación, comprobando que las aplicaciones tienen autorización para acceder a una cola o a un gestor de colas. También puede controlar el uso del API por parte de las aplicaciones, autenticando las llamadas API individuales o incluso los parámetros que estas utilizan.

Flexibilidad
Para mayor flexibilidad, podrá responder a los rápidos cambios que surgen en el entorno de su empresa sin modificar las aplicaciones que confían en los datos de dicho entorno. Por ejemplo, se podrían tener salidas de API que respondiesen a cambios en los tipos de interés, en los tipos de cambio de divisas o en el precio de los componentes en un entorno de fabricación.

Supervisión del uso de una cola o un gestor de colas
En cuanto a la supervisión del uso de una cola o un gestor de colas, puede rastrear el flujo de aplicaciones y mensajes, anotar los errores en las llamadas API, establecer colas de seguimiento a efectos de contabilidad, o bien recopilar estadísticas de uso a efectos de planificación.

¿Qué ocurre cuando se ejecuta una salida de API?

Una vez que haya escrito un programa de salida y lo haya identificado en IBM MQ, el gestor de colas invoca automáticamente el código de salida en los puntos registrados.

Las rutinas de salida de API que se van a ejecutar se identifican en stanzas en Multiplatforms. En este tema se cubren las stanzas de los archivos de configuración `mqs.ini` y `qm.ini`.

La definición de las rutinas puede realizarse en tres lugares:

1. ApiExitCommon, en el archivo mqs.ini, identifica las rutinas, para todo el IBM MQ, aplicadas cuando se inician los gestores de colas. Estos se puede alterar temporalmente mediante rutinas para gestores de colas individuales (consulte el elemento “3” en la página 978 de esta lista).
2. La plantilla ApiExit, en el archivo mqs.ini , identifica rutinas, para todo IBM MQ, copiadas en el conjunto local ApiExit(consulte el elemento “3” en la página 978 de esta lista) cuando se crea un nuevo gestor de colas.
3. ApiExitLocal, en el archivo qm.ini, identifica las rutinas que se aplican a un gestor de colas determinado.

Cuando se crea un gestor de colas, las definiciones de ApiExitTemplate en mqs.ini se copian en las definiciones de ApiExitLocal del archivo qm.ini del nuevo gestor de colas. Cuando se inicia un gestor de colas, se utilizan las definiciones de ApiExitCommon y ApiExitLocal. Las definiciones de ApiExitLocal sustituyen a las definiciones de ApiExitCommon si ambas identifican una rutina con el mismo nombre. El atributo `Sequence` , descrito en “Configurar salidas de API” en la página 983 determina el orden en el que se ejecutan las rutinas definidas en las stanzas.

Utilización de salidas de API en varias instalaciones de IBM MQ

Asegúrese de que las salidas de API escritas para la versión anterior de IBM MQ se usen para funcionar con todas las versiones, porque puede que los cambios efectuados a las salidas en IBM WebSphere MQ 7.1 no funcionen en una versión anterior. Para obtener más información sobre los cambios realizados en las salidas, consulte “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960.

Los ejemplos proporcionados para las salidas de API amqsaem y amqsaxe reflejan los cambios necesarios al escribir salidas. La aplicación cliente debe asegurarse de que las bibliotecas IBM MQ correctas que corresponden a la instalación del gestor de colas con el que está asociada la aplicación están enlazadas con ella antes de iniciar la aplicación.

Multi ***Desarrollo de una salida de API***

Se pueden escribir salidas en C para cada llamada de API.

Salidas disponibles

Hay salidas disponibles para cada llamada de API, como se indica a continuación:

- MQCB, para anular el registro de una devolución de llamada del manejador de objetos especificado y controlar los cambios en la devolución de llamada
- MQCTL, para realizar acciones de control en los manejadores de objetos abiertos para una conexión.
- MQCONN/MQCONNX, para proporcionar un manejador de conexión de gestor de colas que puede utilizarse en llamadas de API posteriores.
- MQDISC, para desconectar de un gestor de colas.
- MQBEGIN, para iniciar una unidad de trabajo (UOW) global.
- MQBACK, para restituir una UOW.
- MQCMIT, para confirmar una UOW.
- MQOPEN, para abrir un recurso IBM MQ para accesos posteriores
- MQCLOSE, para cerrar un recurso IBM MQ que se haya abierto previamente para el acceso
- MQGET, para recuperar un mensaje de una cola abierta previamente para el acceso.
- MQPUT1, para colocar un mensaje en una cola.
- MQPUT, para colocar un mensaje en una cola abierta previamente para el acceso.
- MQINQ, para consultar sobre los atributos de un recurso IBM MQ que se ha abierto previamente para el acceso
- MQSET, para establecer los atributos de una cola abierta previamente para el acceso.
- MQSTAT, para recuperar información de estado.
- MQSUB, para registrar la suscripción de aplicaciones a un tema determinado.
- MQSUBRQ, para realizar una solicitud de suscripción

MQ_CALLBACK_EXIT proporciona una función de salida para realizar antes y después del proceso de devolución de llamada. Para obtener más información, consulte Devolución de llamada - MQ_CALLBACK_EXIT.

Desarrollo de una salida de API

Dentro de las salidas de API, las llamadas tienen el formato siguiente:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

donde *call* es el nombre de llamada MQI sin el prefijo MQ ; por ejemplo, PUT, GET. Los *parameters* controlan la función de la salida, principalmente proporcionando comunicación entre la salida y los bloques de control externo MQAXP (la estructura de parámetros de salida de API) y MQAXC (la estructura de contexto de salida de API). *context* describe el contexto en el que se ha llamado a la salida de API y *ApiCallParameters* representa los parámetros de la llamada MQI.

Para ayudarle a escribir la salida de API, se proporciona una salida de ejemplo, amqsaxe0.c; esta salida genera entradas de rastreo en el archivo que se especifique. Puede utilizar este ejemplo como punto de partida cuando escriba salidas. Para obtener más información sobre la utilización de la salida de ejemplo, consulte “El programa de ejemplo de salida de API” en la página 1103.

Para obtener más información sobre las llamadas de salida de API, bloques de control externos y temas asociados, consulte Referencia de salidas de API.

Para obtener información general sobre cómo escribir, compilar y configurar una salida, consulte “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960.

Utilización de manejadores de mensajes en salidas de API

Se pueden controlar las propiedades de mensaje a las que tiene acceso una salida de API. Las propiedades están asociadas a un ExitMsgHandle. Las propiedades establecidas en una salida de colocación se establecen en el mensaje que se está colocando, pero las propiedades recuperadas en una salida de obtención no se devuelven a la aplicación.

Cuando se registra una función de salida MQ_INIT_EXIT usando una llamada MQXEP con **Function** establecido a MQXF_INIT y **ExitReason** establecido a MQXR_CONNECTION, hay que pasar una estructura MQXEPO como parámetro **ExitOpts**. La estructura MQXEPO contiene el campo ExitProperties, que especifica el conjunto de propiedades que se pone a disposición de la salida. Se especifica como una cadena de caracteres que representan el prefijo de las propiedades, que se corresponde con un nombre de carpeta MQRFH2.

Cada salida de API recibe una estructura MQAXP que contiene un campo ExitMsgHandle. Este campo se establece en un valor generado por IBM MQ y es específico para una conexión. Por consiguiente, el descriptor permanece sin modificarse entre salidas de API del mismo o de diferentes tipos en la misma conexión.

En una salida MQ_PUT_EXIT o MQ_PUT1_EXIT con una **ExitReason** MQXR_BEFORE, es decir, una salida de API realizada antes de colocar un mensaje, cualquier propiedad (distinta de las propiedades del descriptor de mensaje) asociada a ExitMsgHandle al completarse la salida se establece en mensaje que se está colocando. Para evitar que ocurra esto, establezca ExitMsgHandle a MQHM_NONE. También se puede suministrar un descriptor de mensaje distinto..

En MQ_GET_EXIT y MQ_CALLBACK_EXIT, el manejador ExitMsgse borra de las propiedades y se llena con las propiedades especificadas en el campo ExitProperties cuando se ha registrado MQ_INIT_EXIT, que no son las propiedades del descriptor de mensaje. Estas propiedades no están disponibles a la aplicación que lleva a cabo la obtención. Si la aplicación de obtención ha especificado un descriptor de mensaje en el campo MQGMO (opciones del mensaje de obtención), las propiedades asociadas a dicho descriptor de contexto, incluidas las propiedades del descriptor de mensaje, estarán disponibles a la salida de API. Para evitar que ExitMsgHandle se rellene con propiedades, establézcalo a MQHM_NONE.

Nota: Para que las propiedades del mensaje de salida se procesen en:

- Después de la función MQ_GET_EXIT, debe definir una función MQ_GET_EXIT anterior para la salida.
- Antes de la función MQ_CALLBACK_EXIT, debe definir una función MQ_CB_EXIT anterior para la salida.

V 9.3.2 A partir de IBM MQ 9.3.2, las sentencias anteriores relativas a MQ-GET-EXIT y MQ_CALLBACK_EXIT ya no se aplican.

Se proporciona un programa de ejemplo, amqsaem0.c, para ilustrar el uso de los manejadores de mensajes en una salida de API.

Referencia relacionada

Referencia de salidas de usuarios, salidas de API y servicios instalables

Multi *Compilación de salidas de API*

Una vez escrita una salida, se compila y enlaza de la forma siguiente.

Los ejemplos siguientes muestran los comandos que se utilizan para el programa de ejemplo descrito en “El programa de ejemplo de salida de API” en la página 1103. En plataformas distintas de los sistemas Windows, se puede encontrar el código de la salida de API de ejemplo en *MQ_INSTALLATION_PATH*/samp y la biblioteca compartida y enlazada en *MQ_INSTALLATION_PATH*/samp/bin.

Windows Para los sistemas Windows, puede encontrar el código de salida de API de ejemplo en *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* representa el directorio en el que se ha instalado IBM MQ.

Nota: Las directrices sobre la programación de aplicaciones de 64 bits se listan en Estándares de codificación en plataformas de 64 bits

Para clientes de multidifusión, las salidas de API y las salidas de conversión de datos se deben poder ejecutar en el lado del cliente porque es posible que algunos mensajes no pasen por el gestor de colas. Las siguientes bibliotecas forman parte de los paquetes de cliente así como de los paquetes de servidor:

Tabla 141. Bibliotecas incluidas en los paquetes de cliente y servidor	
Sistema operativo	**Bibliotecas**
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bits & 64 bits: libmqm.so & libmqm_r.so
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb

Linux AIX *Compilación de salidas de API en sistemas AIX and Linux*

Ejemplos de cómo compilar salidas de API en sistemas AIX and Linux.

En todas las plataformas, el punto de entrada al módulo es MQStart.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

En AIX

AIX

Compile el código fuente de la salida de API emitiendo uno de los mandatos siguientes:

Aplicaciones de 32 bits

Sin hebras

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

En Linux

Linux

Compile el código fuente de la salida de API emitiendo uno de los mandatos siguientes:

Aplicaciones de 31 bits

Sin hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 32 bits

Sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Compilación de salidas de API en sistemas Windows*

Compile y enlace el programa de salida de API de ejemplo, `amqsaxe0.c`, en Windows

Un archivo de manifiesto es un documento XML opcional que contiene la versión, o cualquier otra información, que se puede incorporar en una aplicación compilada o una DLL.

Si no tiene ningún documento de este tipo, omita el parámetro -manifest *manifest.file* en el mandato **mt**.

Adapte los mandatos en los ejemplos de Figura 101 en la página 983 o Figura 102 en la página 983 para compilar y enlazar `amqsaxe0.c` en Windows. Los mandatos funcionan con Microsoft Visual Studio 2008, 2010 o 2012. En los ejemplos se presupone que el directorio `C:\Archivos de programa\IBM\MQ\tools\c\samples` es el directorio actual.

32 bits

```
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
```

Figura 101. Compile y enlace `amqsaxe0.c` en Windows de 32-bits

64 bits

```
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
```

Figura 102. Compile y enlace `amqsaxe0.c` en Windows de 64 bits

Conceptos relacionados

"El programa de ejemplo de salida de API" en la página 1103
La salida de API de ejemplo genera un rastreo MQI en un archivo especificado por el usuario con un prefijo definido en la variable de entorno **MQAPI_TRACE_LOGFILE** .

IBM i *Compilación de salidas de API en IBM i*
Compilación de salidas de API en IBM i.

Una salida se crea de la siguiente manera (para un ejemplo de lenguaje C):

1. Cree un módulo utilizando CRTCMOD. Compílelo para utilizar teraespacio incluyendo el parámetro `TERASPACE(*YES *TSIFC)`.
2. Cree un programa de servicio en el módulo utilizando CRTSRVPGM. Debe enlazarlo con el programa de servicio QMQM/LIBMQMZF_R para las salidas de API de varias hebras.

Configurar salidas de API

Debe configurar IBM MQ para habilitar las salidas de API cambiando la información de configuración.

Para cambiar la información de configuración, debe cambiar las stanzas que definen las rutinas de salida y la secuencia en la que se ejecutan. Esta información se puede modificar de las siguientes formas:

- Windows Linux Utilización de IBM MQ Explorer en Windows y Linux (plataformas x86 y x86-64).
- Windows Utilización del mandato **amqmdain** en Windows.
- Multi Utilización de los archivos `mqs.ini` y `qm.ini` directamente en Multiplatforms.

El archivo `mqs.ini` contiene información relacionada con todos los gestores de colas de un nodo determinado. Lo puede encontrar en las ubicaciones siguientes:

- Linux AIX En el directorio `/var/mqm` en AIX and Linux.
- Windows En la WorkPath especificada en la clave `HKLM\SOFTWARE\IBM\WebSphere MQ` en sistemas Windows .
- IBM i En el directorio `/QIBM/UserData/mqm` en IBM i.

El archivo `qm.ini` contiene información relevante para un gestor de colas específico. Hay un archivo de configuración de gestor de colas para cada gestor de colas en la raíz del árbol de directorios que ocupa el gestor de colas. Por ejemplo, la vía de acceso y el nombre de un archivo de configuración para un gestor de colas llamado QMNAME es:

IBM i En sistemas IBM i:

```
/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
```

Linux AIX En sistemas AIX and Linux:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

Windows En sistemas Windows:

```
C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
```

Antes de editar un archivo de configuración, haga una copia de seguridad a fin de tener una copia del archivo por si la necesita.

Puede editar los archivos de configuración de cualquiera de las formas siguientes:

- Automáticamente, utilizando mandatos que modifiquen la configuración de gestores de colas en el nodo.
- Manualmente, utilizando un editor de texto estándar.

Si establece un valor incorrecto en un atributo del archivo de configuración, el valor se ignora y se emite un mensaje de operador para indicar el problema. El efecto es el mismo que perder el atributo por completo.

Stanzas para configurar

Las stanzas que hay que cambiar son las siguientes:

ApiExitCommon

Definida en `mqs.ini` y en IBM MQ Explorer en la página de propiedades de IBM MQ, debajo de Salidas.

Cuando se inicia cualquier gestor de colas, se leen los atributos de esta stanza y, a continuación, se sustituyen por las salidas de API definidas en el archivo `qm.ini`.

ApiExitTemplate

Definida en `mqs.ini` y en IBM MQ Explorer en la página de propiedades de IBM MQ, debajo de Salidas.

Cuando se crea un gestor de colas, los atributos de esta stanza se copian en el archivo qm.ini que se acaba de crear bajo la stanza ApiExitLocal.

ApiExitLocal

Definida en `qm.ini` y en IBM MQ Explorer en la página de propiedades del gestor de colas, debajo de Salidas.

Cuando se inicia el gestor de colas, las salidas de API definidas aquí alteran temporalmente los valores predeterminados definidos en el archivo `mqs.ini`.

Atributos de las stanzas

- Nombre la salida de API utilizando los atributos siguientes:

  Name=nombre_ApiExit
  El nombre que describe la salida de API que se ha pasado en el campo ExitInfoName de la estructura MQAXP.

  Este nombre debe ser exclusivo, con un máximo de 48 caracteres de longitud y sólo puede contener caracteres válidos para los nombres de objetos IBM MQ (por ejemplo, nombres de colas).

- Identificar el módulo y el punto de entrada del código de salida de la API para ejecutar utilizando los atributos siguientes:

  Function=nombre_función
  El nombre del punto de entrada de la función al módulo que contiene el código de la salida de API. Este punto de entrada es la función MQ_INIT_EXIT.

  La longitud de este campo está limitada a MQ_EXIT_NAME_LENGTH.

  Module=nombre_módulo
  El módulo que contiene el código de la salida de API.

  Si este campo contiene el nombre de vía de acceso completo del módulo, se utilizará tal y como está.

  Si este campo contiene sólo el nombre de módulo, el módulo se encuentra utilizando el atributo `ExitsDefaultPath` en `ExitPath` en qm.ini.

  En plataformas que dan soporte a bibliotecas con hebras independientes, debe proporcionar tanto una versión sin hebras como una versión con hebras del módulo de salida de API. La versión con hebras debe tener un sufijo `_r`. La versión con hebras del apéndice de aplicación de IBM MQ añade implícitamente un sufijo `_r` al nombre de módulo especificado antes de cargarlo.

  La longitud de este campo está limitada a la longitud máxima de vía de acceso a la que dé soporte la plataforma.

- Pasar opcionalmente los datos con la salida utilizando el atributo siguiente:

  Data=nombre_datos
  Los datos que se han de pasar a la salida de API en el campo ExitData de la estructura MQAXP.

  Si incluye este atributo, se suprimirán los espacios en blanco iniciales y de cola, la serie restante se truncará a 32 caracteres y el resultado se pasará a la salida. Si omite este atributo, se pasará el valor predeterminado de 32 espacios en blanco.

  La longitud máxima de este campo es de 32 caracteres.

- Identificar la secuencia de esta salida en relación con otras salidas utilizando el atributo siguiente:

  Sequence=número_secuencia
  La secuencia en que se llama a esta salida de API es relativa para las otras salidas de API. Se llama antes a una salida con un número de secuencia bajo que a una salida con un número de secuencia más alto. No es necesario que los números de secuencia de las salidas sean contiguos. Una secuencia de 1, 2, 3 tiene el mismo resultado que una secuencia de 7, 42, 1096. Si dos salidas tienen el mismo número de secuencia, el gestor de colas decide a cuál de ellos llamará en primer lugar. Puede saber a cuál se ha llamado después del suceso, colocando la hora o un marcador en ExitChainArea, que se indica mediante ExitChainAreaPtr en MQAXP, o escribiendo su propio archivo de anotaciones.

Este atributo es un valor numérico sin signo.

Stanzas de ejemplo

El archivo mqs.ini de ejemplo contiene las stanzas siguientes:

ApiExitTemplate
Esta stanza define una salida con el nombre descriptivo `OurPayrollQueueAuditor`, el nombre de módulo `auditory` el número de secuencia 2. Se pasa un valor de datos de 123 a la salida.

ApiExitCommon
Esta stanza define una salida con el nombre descriptivo `MQPoliceman`, el nombre de módulo `tmqpy` el número de secuencia 1. Los datos pasados son una instrucción (`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
```

El archivo de ejemplo qm.ini siguiente contiene una definición ApiExitLocal de una salida con el nombre descriptivo `ClientApplicationAPIchecker`, nombre de módulo `ClientAppChecker` y número de secuencia 3.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

Programas de salida de canal para canales de mensajes

Esta colección de temas contiene información sobre los programas de salida de canal de IBM MQ para los canales de mensajería.

Los agentes de canal de mensajes (MCA) también pueden invocar salidas de conversión de datos. Para obtener más información sobre la escritura de salidas de conversión de datos, consulte “Escribir salidas de conversión de datos” en la página 1009.

Parte de esta información también es aplicable a las salidas para canales MQI, que conectan IBM MQ MQI clients con gestores de colas. Para obtener más información, consulte Programas de salida de canal para canales MQI.

Los programas de salida de canal se invocan en lugares definidos del proceso ejecutado por los programas de MCA.

Algunos de estos programas de salida de usuario funcionan en parejas complementarias. Por ejemplo, si el MCA emisor invoca un programa de salida de usuario para cifrar mensajes para su transmisión, el proceso complementario debe estar funcionando en el extremo receptor para invertir el proceso.

La Tabla 142 en la página 987 muestra los tipos de salida de canal que están disponibles para cada tipo de canal.

Tabla 142. Salidas de canal disponibles para cada tipo de canal

Tipo de canal	Salida de mensajes	Salida de reintento de mensaje	Salida de recepción	Salida de seguridad	Salida de emisión	Salida de definición automática
Canal emisor	Sí		Sí	Sí	Sí	
Canal servidor	Sí		Sí	Sí	Sí	
Canal emisor de clúster	Sí		Sí	Sí	Sí	Sí
Canal receptor	Sí	Sí	Sí	Sí	Sí	Sí
Canal peticionario	Sí	Sí	Sí	Sí	Sí	
Canal receptor de clúster	Sí	Sí	Sí	Sí	Sí	Sí
Canal de conexión de cliente			Sí	Sí	Sí	
Canal de conexión de servidor			Sí	Sí	Sí	Sí

Notas: z/OS

1. En z/OS, la salida de definición automática es aplicable sólo a los canales emisor de clúster y receptor de clúster.

Si piensa ejecutar salidas de canal en un cliente, no puede utilizar la variable de entorno MQSERVER. En lugar de ello, cree y especifique una tabla de definiciones de canales de cliente (CCDT) tal como se describe en Tabla de definiciones de canales de cliente.

Visión general del proceso

Una visión general de cómo utilizan los MCA los programas de salida de canal.

Durante el arranque, los MCA intercambian un diálogo de arranque para sincronizar el proceso. A continuación, pasan a un intercambio de datos que incluye las salidas de seguridad. Estas salidas deben finalizar correctamente para que se complete la fase de arranque y se permita la transferencia de mensajes.

La fase de comprobación de seguridad es un bucle, como se muestra en la Figura 103 en la página 988.

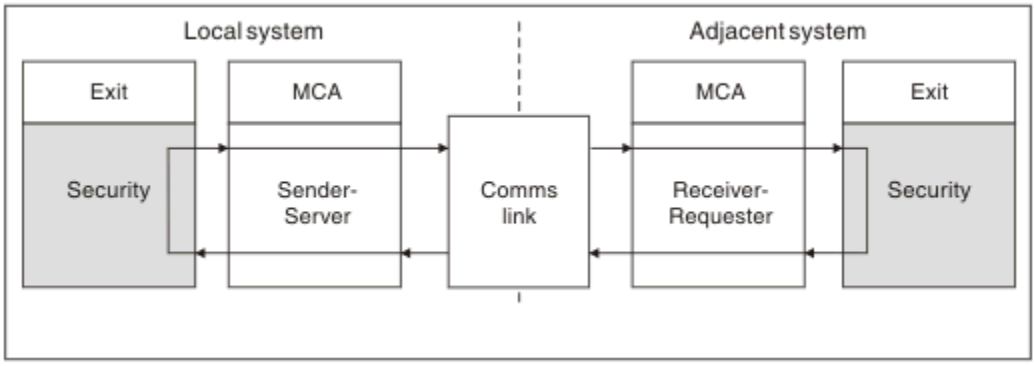


Figura 103. Bucle de salida de seguridad

Durante la fase de transferencia de mensaje, el MCA emisor obtiene mensajes de una cola de transmisión, invoca la salida de mensaje, invoca la salida de envío y, a continuación, envía el mensaje al MCA receptor, como se muestra en la Figura 104 en la página 988.

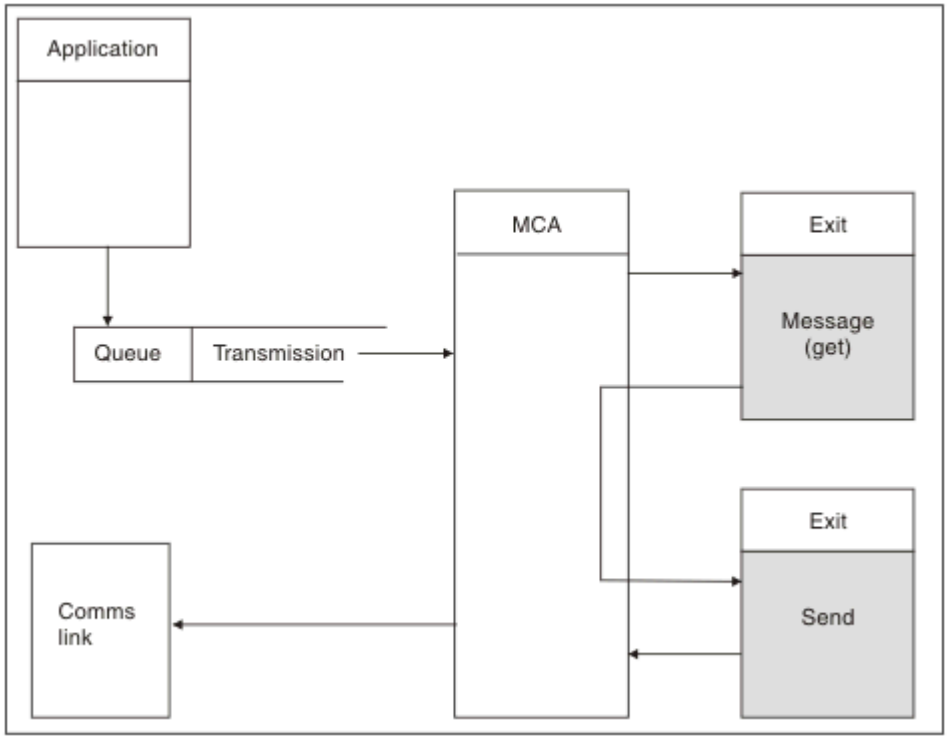


Figura 104. Ejemplo de una salida de envío en el extremo emisor del canal de mensajes

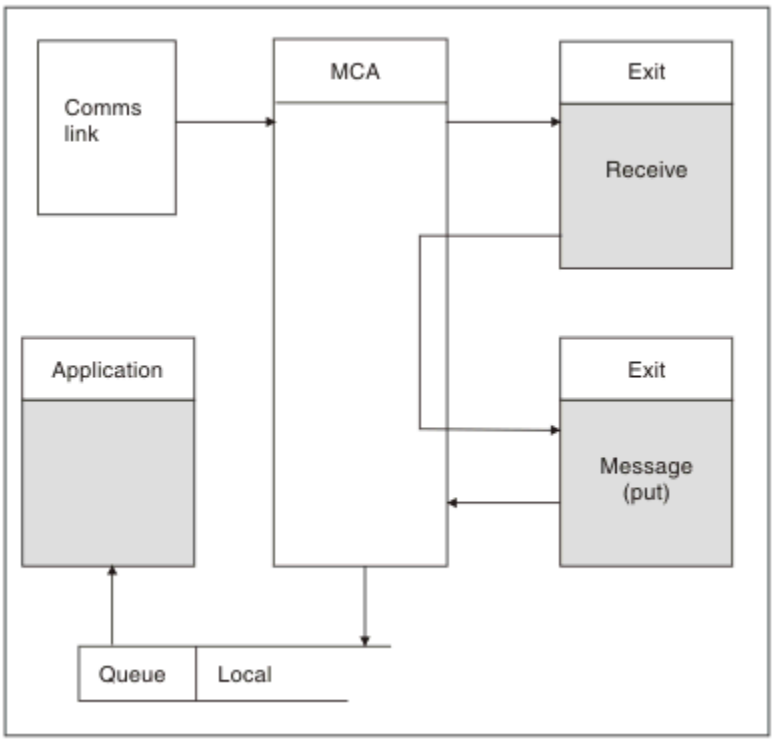


Figura 105. Ejemplo de una salida de recepción en el extremo receptor del canal de mensajes

El MCA recibe el mensaje desde el enlace de comunicaciones, invoca la salida de recepción, invoca la salida de mensaje y, a continuación, transfiere el mensaje a la cola local, como se muestra en la Figura 105 en la página 989. La salida de recepción se puede invocar más de una vez antes de invocar la salida de mensaje.

Cómo escribir programas de salida de canal

Puede utilizar la información siguiente para ayudarle a escribir programas de salida de canal.

Las salidas de usuario y los programas de salida de canal pueden utilizar todas las llamadas MQI, excepto si se indica lo contrario en las secciones que siguen. Para MQ V7 y posteriores, la estructura de MQCXP versión 7 y posteriores contiene el manejador de conexión hConn, que puede utilizarse en lugar de emitir MQCONN. En las versiones anteriores, para obtener el manejador de conexión debe emitirse una llamada MQCONN, aunque se devuelve un aviso MQRC_ALREADY_CONNECTED porque el canal está conectado con el gestor de colas.

Tenga en cuenta que la salida de canal deben ser de hebra protegida.

Para las salidas de los canales de conexión con el cliente, el gestor de colas al que intenta conectarse la salida depende de cómo se ha vinculado la salida. Si la salida se enlazó con MQM.LIB (o QMQM/LIBMQM en IBM i) y no especifica un nombre de gestor de colas en la llamada MQCONN, la salida intenta conectar con el gestor de colas predeterminado del sistema. Si la salida se enlazó con MQM.LIB (o QMQM/LIBMQM en IBM i) y especifica el nombre del gestor de colas que se pasó a la salida a través del campo QMgrName de MQCD, la salida intenta conectar con dicho gestor de colas. Si la salida se ha vinculado con MQIC.LIB o cualquier otra biblioteca, la llamada MQCONN falla tanto si se especifica un nombre de gestor de colas como si no.

Debe evitar modificar el estado de la transacción asociada con el parámetro Hconn que se ha pasado en una salida de canal; no debe utilizar los verbos MQCMIT, MQBACK o MQDISC con el parámetro Hconn del canal y no puede utilizar el verbo MQBEGIN especificando el parámetro Hconn del canal.

Si se utiliza MQCONNX especificando MQCNO_HANDLE_SHARE_BLOCK o MQCNO_HANDLE_SHARE_NO_BLOCK para crear una nueva conexión de IBM MQ, deberá asegurarse de que la conexión se ha gestionado correctamente y se desconecta del gestor de colas correctamente. Por

ejemplo, una salida de canal que cree una conexión nueva con el gestor de colas en cada invocación sin desconectarse, tendrá como consecuencia que los manejadores de conexión se acumularán y aumentará el número de hebras de agente.

Una salida se ejecuta en la misma hebra que el MCA y utiliza el mismo manejador de conexión. Por lo tanto, se ejecuta dentro de la misma UOW que el MCA y el canal confirma o restituye todas las llamadas realizadas bajo el punto de sincronización en el extremo del lote.

Por lo tanto, una salida de mensajes de canal puede enviar mensajes de notificación que sólo se comprometen con esta cola cuando el lote que contiene el mensaje original se confirma. Por lo tanto, es posible emitir llamadas MQI de punto de sincronización desde una salida de mensajes de canal.

Una salida de canal pueden cambiar los campos del MQCD. Sin embargo, estos cambios no se aplican, excepto en las circunstancias que se enumeran. Si un programa de salida de canal cambia un campo de la estructura de datos MQCD, el proceso del canal IBM MQ hace caso omiso del nuevo valor. Sin embargo, el nuevo valor permanece en el MQCD y se le pasa a las salidas restantes en una cadena de salida y a cualquier conversación que comparta la instancia de canal. Para obtener más información, consulte Cambio de campos MQCD en una salida de canal

Además, para programas escritos en C, no se debe utilizar una función de biblioteca C no reentrante en un programa de salida de canal.

Linux AIX Si utiliza varias bibliotecas de salida de canal simultáneamente, pueden surgir problemas en algunas plataformas UNIX and Linux si el código de dos salidas diferentes contiene funciones que se llaman exactamente igual. Cuando se carga una salida de canal, el cargador dinámico resuelve los nombres de función de la biblioteca de salida para las direcciones donde se carga la biblioteca. Si dos bibliotecas de salida definen funciones diferentes que se llaman exactamente igual, este proceso de resolución puede resolver incorrectamente los nombres de función de una biblioteca de modo que utilizarán las funciones de la otra. Si ocurre este problema, indique al enlazador que sólo debe exportar la salida y las funciones MQStart necesarias, ya que estas funciones no se ven afectadas. Las otras funciones deben tener visibilidad local para que no las utilicen las funciones de fuera de su propia biblioteca de salida. Consulte la documentación del enlazador para obtener más información.

Todas las salidas se invocan con una estructura de parámetros de salida de canal (MQCXP), una estructura de definición de canal (MQCD), un almacenamiento intermedio de datos preparados, el parámetro de longitud de datos y el parámetro de longitud del almacenamiento intermedio. No debe superarse la longitud del almacenamiento intermedio:

- Para las salidas de mensajes, debe tener en cuenta el mensaje de mayor tamaño que debe enviarse a través del canal más la longitud de la estructura MQXQH.
- Para enviar y recibir salidas, el tamaño máximo que debe permitirse para el almacenamiento intermedio es el siguiente:

  LU6.2
  32 KB

  TCP:

  IBM i IBM i 16 KB

  IBM i Otros 32 KB

  Nota: La longitud máxima utilizable sería 2 bytes menor que esta longitud. Compruebe el valor devuelto en MaxSegmentLength para obtener más detalles. Para obtener más información, consulte MaxSegmentLength.

  NetBIOS:
  64 KB

  SPX:
  64 KB

  Nota: Las salidas de recepción en los canales emisores y las salidas del remitente en los canales receptores utilizan almacenamientos intermedios de 2 KB para TCP.

- Para las salidas de seguridad, el recurso de gestión de colas distribuidas asigna un almacenamiento intermedio de 4000 bytes.

Es aceptable que la salida devuelva un almacenamiento intermedio alternativo, junto con los parámetros relevantes. Consulte "Programas de salida de canal para canales de mensajes" en la página 986 para obtener detalles de la llamada.

z/OS *Escritura de programas de salida de canal en z/OS*

Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para z/OS.

Las salidas se inician como si se realizarán mediante un LINK de z/OS en:

- Estado de programa anómalo no autorizado
- Modalidad de control de espacio de direcciones primaria
- Modalidad de memoria no cruzada
- Modalidad de registro sin acceso
- Modalidad de direccionamiento de 31 bits

Los módulos editados por enlace se deben colocar en el conjunto de datos especificado en la sentencia CSQXLIB DD del procedimiento del espacio de direcciones del iniciador de canal. Los nombres de los módulos de carga se especifican como los nombres de salida en la definición de canal.

Cuando se escriben salidas de canal para z/OS, se aplican las reglas siguientes:

- Las salidas se deben escribir en el ensamblador o en C. Si se utiliza C, debe ser compatible con el entorno de programación de sistemas C para las salidas del sistema, como se describe en la publicación z/OS C/C++ Guía de programación.
- Las salidas se cargan desde las bibliotecas no autorizadas definidas mediante una CSQXLIB DD. Siempre y cuando CSQXLIB tenga DISP=SHR, se pueden actualizar las salidas mientras se ejecuta el iniciador de canal. La nueva versión se utiliza cuando se reinicia el canal.
- Las salidas deben volver a entrar y deben poder ejecutarse en cualquier lugar del almacenamiento virtual.
- Cuando regresan, las salidas deben restablecer el entorno como estaba al entrar.
- Las salidas deben liberar cualquier almacenamiento obtenido o asegurarse de que está libre invocando una salida posterior.

  En el caso de que el almacenamiento debe persistir entre invocaciones, utilice el servicio STORAGE de z/OS o la función de biblioteca4kmalc para la programación del sistema C.

  Para obtener más información acerca de esta función, consulte 4kmalc() -- Asignar almacenamiento alineado por página.
- Se pueden utilizar todas las llamadas MQI de IBM MQ MQI, salvo MQCMIT o CSQBCMT y MQBACK o CSQBBAK. Deben estar contenidas después de MQCONN (con un nombre de gestor de colas en blanco). Si se utilizan estas llamadas, la salida debe estar editada mediante enlace con el apéndice CSQXSTUB.

  La excepción a esta regla es que las salidas del canal de seguridad pueden emitir llamadas MQI de confirmación y restitución. Para emitir estas llamadas, codifique los verbos CSQXCMT y CSQXBAK, en lugar de MQCMIT o CSQBCMT y MQBACK o CSQBBAK.
- Todas las salidas que utilizan el apéndice CSQXSTUB de IBM WebSphere MQ 7.0 o posterior, deben editarse mediante enlace en una biblioteca de carga CSQXLIB con formato PDS-E.
- Las salidas no deben utilizar ningún servicio del sistema que genere una espera, ya que utilizar los servicios del sistema puede afectar negativamente algunos o todos los otros canales. Normalmente, muchos canales se ejecutan bajo un TCB único. Si realiza algo en una salida que genera una espera y no utiliza MQXWAIT, todos estos canales tendrán que esperar. Hacer que los canales tengan que esperar no genera problemas de funcionamiento pero puede tener un efecto adverso en el rendimiento. La mayor parte de los SVC conllevan esperas, por lo que debe evitarlos, a excepción de los SVC siguientes:
  - GETMAIN/FREEMAIN/STORAGE

– LOAD/DELETE

Por lo tanto, en general, evite SVC, PCs y E/S. En su lugar, utilice la llamada MQXWAIT.

- Las salidas no emiten ESTAE o SPIE, excepto en cualquier subtarea asociada, debido a que su manejo de errores puede interferir con el manejo de errores que realiza IBM MQ. Esto significa que es posible que IBM MQ no pueda recuperarse de un error o que su programa de salida no reciba toda la información sobre el error.
- La llamada MQXWAIT (consulte MQXWAIT) proporciona un servicio de espera que espera a sucesos de E/S y otros sucesos. Si se utiliza este servicio, las salidas no deben utilizar la pila de enlace.

  En el caso de E/S y otras funciones que no proporcionan recursos de no bloqueo o un ECB que esperar, se debe asociar, ATTACH, una subtarea separada y se debe esperar a que termine mediante MQXWAIT. Debido a los procesos que implica esta técnica, solo debe utilizarse este recurso en la salida de seguridad.

- La llamada MQI MQDISC no genera una confirmación implícita en el programa de salida. Una confirmación del proceso de canal solo se realiza cuando lo dicta el protocolo del canal.

Se proporcionan los siguientes ejemplos de salidas con IBM MQ for z/OS:

CSQ4BAX0
Este ejemplo está escrito en el ensamblador e ilustra el uso de MQXWAIT.

CSQ4BCX1 y CSQ4BCX2
Estos ejemplos están escritos en C e ilustran cómo acceder a los parámetros.

CSQ4BCX3 y CSQ4BAX3
Estos ejemplos están escritos en C y en el ensamblador respectivamente.

El ejemplo CSQ4BCX3, precompilado en SCSQAUTH LOADLIB, debe funcionar en la propia salida sin que sean necesarios cambios. Puede crear una LOADLIB, por ejemplo, con el nombre MY.TEST.LOADLIB, y copiar el miembro SCSQAUTH(CSQ4BCX3) en la misma.

Para configurar una salida de seguridad en una conexión de cliente, realice el procedimiento siguiente:

1. Establezca un segmento OMVS válido para el ID de usuario que utiliza el iniciador de canal.

   Esto permite al iniciador de canal de IBM MQ for z/OS utilizar TCP/IP con la interfaz de socket de z/OS UNIX System Services (z/OS UNIX), para facilitar el proceso de salida. Tenga en cuenta que no es necesario definir un segmento OMVS para el ID de usuario de cualquier cliente que se conecte.
2. Asegúrese de que el propio de código de salida solo se ejecute en un entorno controlado por programa.

   Esto significa que todo lo que se cargue en el espacio de direcciones CHINIT se debe cargar desde una biblioteca controlada por programa, lo que significa todas las bibliotecas de STEPLIB, y cualquier biblioteca mencionada en CSQXLIB y

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   Para establecer una biblioteca de carga como programa controlado, utilice un mandato similar a este ejemplo:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   A continuación, puede activar o renovar el entorno controlado por programa emitiendo el mandato:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

3. Añada la salida LOADLIB a CSQXLIB DD, en el procedimiento iniciado CHINIT, emitiendo el mandato siguiente:

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

   Esto activa la salida para el canal especificado.
4. Su gestor de seguridad externo (ESM) lista cualquier otra biblioteca que deba estar controlada por programa, pero tenga en cuenta que ninguna de las bibliotecas ESM o C necesita bajo control de programa.

Consulte Canal de conexión de servidor IBM MQ for z/OS para obtener más información sobre cómo configurar una salida de seguridad utilizando el ejemplo CSQ4BCX3.

CSQ4BCX4
Este ejemplo está escrito en C y muestra cómo utilizar los campos **RemoteProduct** y **RemoteVersion** en MQCXP.

Conceptos relacionados

“Escritura de programas de salida de canal en IBM i” en la página 993
Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para IBM i.

“Escritura de programas de salida de canal en AIX, Linux, and Windows” en la página 994
Puede utilizar la información siguiente como ayuda a escribir programas de salida de canal para sistemas AIX, Linux, and Windows.

Referencia relacionada

Canal de conexión del servidor de IBM MQ for z/OS

IBM i *Escritura de programas de salida de canal en IBM i*

Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para IBM i.

La salida es un objeto de programa escrito en el lenguaje ILE C, ILE RPG o ILE COBOL. Los nombres de programa de salida y sus bibliotecas se especifican en la definición de canal.

Tenga en cuenta las siguientes condiciones cuando cree y compile un programa de salida:

- El programa debe ser de proceso múltiple y debe crearse con el compilador ILE C, ILE RPG o ILE COBOL. Para ILE RPG, debe especificar la especificación de control THREAD(*SERIALIZE) y, para ILE COBOL, debe especificar SERIALIZE para la opción THREAD de la sentencia PROCESS. Los programas también deben estar enlazados con las bibliotecas de IBM MQ con hebras: QMQM/LIBMQM_R en el caso de ILE C y ILE RPG, y AMQ0STUB_R en el caso de ILE COBOL. Para obtener información adicional sobre el proceso múltiple de las aplicaciones RPG o COBOL, consulte la Guía del programador correspondiente al lenguaje.
- IBM MQ for IBM i requiere que los programas de salida estén habilitados para el soporte de teraespacio. (El teraespacio es una forma de memoria compartida introducida en OS/400 V4R4). Para los compiladores ILE RPG y COBOL, los programas compilados en OS/400 V4R4 o posteriores también están habilitados. Para C, los programas deben compilarse con las opciones TERASPACE(*YES *TSIFC) especificadas en los mandatos CRTCMOD o CRTBNDC.
- Una salida que devuelve un puntero a su propio espacio de almacenamiento intermedio debe asegurarse de que el objeto al que se apunta existe más allá del intervalo de tiempo del programa de salida de canal. El puntero no puede ser la dirección de una variable en la pila de programas, ni de una variable en el almacenamiento dinámico del programa. En su lugar, el puntero debe obtenerse del sistema. Un ejemplo es un espacio de usuario creado en la salida de usuario. Para asegurarse de que toda área de datos asignada por el programa de salida de canal todavía esté disponible para el MCA cuando finalice el programa, la salida de canal debe ejecutarse en el grupo de activación del emisor o un grupo de activación con nombre. Para ello, establezca el parámetro ACTGRP de CRTPGM en un valor definido por el usuario o *CALLER. Si el programa se crea de esta forma, el programa de salida de canal puede asignar la memoria dinámica y pasar un puntero a esta memoria al MCA.

Conceptos relacionados

“Escritura de programas de salida de canal en AIX, Linux, and Windows” en la página 994
Puede utilizar la información siguiente como ayuda a escribir programas de salida de canal para sistemas AIX, Linux, and Windows.

“Escritura de programas de salida de canal en z/OS” en la página 991
Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para z/OS.

ALW *Escritura de programas de salida de canal en AIX, Linux, and Windows*

Puede utilizar la información siguiente como ayuda a escribir programas de salida de canal para sistemas AIX, Linux, and Windows.

Siga las instrucciones descritas en “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960. Utilice la siguiente información específica de salida de canal, si procede:

La salida se debe escribir en C, y es una DLL en Windows.

Defina una rutina MQStart() ficticia en la salida y especifique `MQStart` como punto de entrada de la biblioteca. Figura 106 en la página 994 muestra cómo definir una entrada en el programa:

```
#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
}
```

Figura 106. Código fuente de ejemplo para una salida de canal

Cuando se escriben salidas de canal para Windows utilizando Visual C++, debe escribir su propio archivo DEF. En Figura 107 en la página 994 se muestra un ejemplo. Para obtener más información sobre cómo escribir programas de salida de canal, consulte “Cómo escribir programas de salida de canal” en la página 989.

```
EXPORTS
ChannelExit
```

Figura 107. Archivo DEF de ejemplo para Windows

Conceptos relacionados

“Escritura de programas de salida de canal en IBM i” en la página 993
Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para IBM i.

“Escritura de programas de salida de canal en z/OS” en la página 991
Puede utilizar la siguiente información como ayuda para escribir y compilar programas de salida de canal para z/OS.

Programas de salida de seguridad de canal

Puede utilizar programas de salida de seguridad para verificar que la aplicación asociada en el otro extremo de un canal es genuina. Esto se conoce como autenticación.

Para especificar que un canal debe utilizar una salida de seguridad, especifique el nombre de salida en el campo **SCYEXIT** de la definición de canal.

Nota: La autenticación también se puede lograr con registros de autenticación del canal. Los Registros de autenticación de canal proporcionan una gran flexibilidad para evitar el acceso a los gestores de colas de determinados usuarios y canales, y para correlacionar usuarios remotos con los identificadores de usuario de IBM MQ. El soporte de TLS también lo proporciona IBM MQ para autenticar a los usuarios y para proporcionar comprobaciones de integridad de datos y cifrado para los datos. Para obtener más información sobre TLS, consulte Protocolos de seguridad TLS en IBM MQ. Sin embargo, si necesita formas más sofisticadas (o diferentes) de proceso de seguridad y otros tipos de comprobaciones y el establecimiento de un contexto de seguridad, le recomendamos que escriba de salidas de seguridad.

Los atributos de DN de sujeto y de emisor aparecen en los siguientes atributos de estado de canal:

- SSLPEER (PCF selector MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF selector MQCACH_SSL_CERT_ISSUER_NAME)

Estos valores son devueltos por los mandatos de estado de canal, así como los datos pasados a las salidas de seguridad de canal que se indican:

- SSLPeerNamePtr de MQCD
- SSLRemCertIssNamePtr de MQCXP

Una salida de seguridad puede escribirse en C o en Java.

Los programas de salida de seguridad de canal se llaman en los siguientes puntos del ciclo de proceso de un MCA:

- Durante el inicio y la finalización del MCA.
- Inmediatamente después de que la negociación de datos inicial finalice en el inicio del canal. El extremo receptor o servidor del canal puede iniciar un intercambio de mensajes de seguridad con el extremo remoto mediante un mensaje que se entrega a la salida de seguridad en el extremo remoto. También puede rechazar hacerlo. El programa de salida se inicia de nuevo para procesar los mensajes de seguridad recibidos desde el extremo remoto.
- Inmediatamente después de que la negociación de datos inicial finalice en el inicio del canal. El extremo emisor o peticionario del canal procesa un mensaje de seguridad recibido del extremo remoto, o inicia un intercambio de seguridad si el extremo remoto no puede hacerlo. El programa de salida se vuelve a iniciar para procesar todos los mensajes de seguridad que pueden haberse recibido posteriormente.

Un canal peticionario nunca se llama con MQXR_INIT_SEC. El canal notifica al servidor que tiene un programa de salida de seguridad y el servidor tiene la oportunidad de iniciar una salida de seguridad. Si no tiene uno, se informa al solicitante y se devuelve un flujo de longitud cero al programa de salida.

Nota: Evite el envío de mensajes de seguridad de longitud cero.

En las figuras Figura 108 en la página 996 a Figura 111 en la página 998 se muestran ejemplos de los datos intercambiados por los programas de salida de seguridad. Estos ejemplos muestran la secuencia de sucesos que se producen y que implica la salida de seguridad del receptor y la salida de seguridad del emisor. Las filas sucesivas de las figuras representan el paso del tiempo. En algunos casos, los sucesos en el receptor y el emisor no están correlacionados y, por lo tanto, pueden producirse al mismo tiempo o en distintos momentos. En otros casos, un suceso en un programa de salida tiene como resultado un suceso complementario que se produce posteriormente en el otro programa de salida. Por ejemplo, en Figura 108 en la página 996:

1. El receptor y el emisor se invocan con MQXR_INIT, pero estas invocaciones no están correlacionadas y, por lo tanto, pueden producirse al mismo tiempo o en distintos momentos.
2. El siguiente receptor se vuelve a invocar con MQXR_INIT_SEC, pero devuelve MQXCC_OK, que no requiere sucesos complementarios en la salida del emisor.
3. A continuación, el emisor se invoca con MQXR_INIT_SEC. Esto no está correlacionado con la invocación del receptor con MQXR_INIT_SEC. El emisor devuelve MQXCC_SEND_SEC_MSG, lo que provoca un suceso complementario en la salida de receptor.
4. A continuación, el receptor se invoca con MQXR_SEC_MSG y devuelve MQXCC_SEND_SEC_MSG, lo que provoca un suceso complementario en la salida del emisor.

5. Seguidamente, el emisor se invoca con MQXR_SEC_MSG y devuelve MQXCC_OK, que no requiere sucesos complementarios en la salida de receptor.

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	

Figura 108. Intercambio con acuerdo iniciado por el emisor

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

Figura 109. Intercambio sin acuerdo iniciado por el emisor

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

Figura 110. Intercambio con acuerdo iniciado por el receptor

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	

Figura 111. Intercambio sin acuerdo iniciado por el receptor

El programa de salida de seguridad de canal se pasa a un almacenamiento intermedio del agente que contiene los datos de seguridad, excluyendo las cabeceras de transmisión generadas por la salida de seguridad. Estos datos pueden ser cualquier tipo de datos adecuados que permita que ambos extremos del canal puedan llevar a cabo la validación de seguridad.

El programa de salida de seguridad en los extremos emisor y receptor del canal de mensajes puede devolver a las llamadas uno de los dos códigos de respuesta siguientes:

- El intercambio de seguridad ha finalizado sin errores
- Suprima el canal y ciérrelo

Nota:

1. Las salidas de seguridad de canal suelen funcionar en pares. Al definir los canales adecuados, asegúrese de que se especifican programas de salida compatibles para ambos extremos del canal.
2. IBM i En IBM i , los programas de salida de seguridad que se han compilado con `Use adopted authority` (USEADPAUT = *YES) pueden adoptar la autorización QMQM o QMQMADM. Tenga en cuenta que la salida no utiliza esta característica porque puede suponer un riesgo de seguridad para el sistema.
3. En un canal TLS en el que el otro extremo del canal proporciona un certificado, la salida de seguridad recibe el Nombre distinguido del sujeto de este certificado en el campo MQCD al que acceden SSLPeerNamePtr y el Nombre distinguido del emisor en el campo MQCXP, al que se accede mediante SSLRemCertIssNamePtr. Este nombre puede servir:
   - para restringir el acceso a través del canal TLS
   - para cambiar MQCD.MCAUserIdentifier en función del nombre.

Conceptos relacionados

Conceptos de TLS (Transport Layer Security)

Referencia relacionada

Registros de autenticación de canal

Desarrollo de una salida de seguridad

Se puede escribir una salida de seguridad utilizando el código esqueleto de salida de seguridad.

Figura 112 en la página 999 ilustra cómo escribir una salida de seguridad.

```
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 */

}
```

Figura 112. Código de esqueleto de una salida de seguridad

Debe existir el punto de entrada estándar de IBM MQ MQStart, pero no es necesario que realice ninguna función. El nombre de la función (EntryPoint en este ejemplo) se puede modificar, pero hay que exportar la función cuando se compile y enlace la biblioteca. Como en el ejemplo anterior, los punteros pChannelExitParms y pChannelDefinition tienen que convertirse (cast) a PMQCXP y PMQCD respectivamente. Para obtener información general sobre la llamada a salidas de canal y el uso de parámetros, consulte MQ_CHANNEL_EXIT. Estos parámetros se utilizan en una salida de seguridad tal como se indica a continuación:

PMQVOID pChannelExitParms
entrada/salida

Puntero a la estructura MQCXP - convertir a PMQCXP para acceder a los campos. Esta estructura se utiliza para comunicar entre la salida y el MCA. Los campos siguientes en MQCXP son de especial interés en una salida de seguridad:

ExitReason
Indica a la salida de seguridad el estado actual del intercambio de seguridad y se utiliza para decidir qué acción hay que emprender.

ExitResponse
Es la respuesta al MCA que determina la etapa siguiente en el intercambio de seguridad.

ExitResponse2
Distintivos de control adicionales para controlar la forma en que el MCA interpreta la respuesta de la salida de seguridad.

ExitUserArea
16 bytes (máximo) de almacenamiento que pueden ser utilizados por la salida de seguridad para mantener el estado entre llamadas.

ExitData
Contiene los datos especificados en el campo SCYDATA de la definición de canal (32 bytes rellenados a la derecha con blancos).

PMQVOID pChannelDefinition
entrada/salida

Puntero a la estructura MQCD - convertir a PMQCD para acceder a los campos. Este parámetro contiene la definición del canal. Los campos siguientes en MQCD son de especial interés en una salida de seguridad:

ChannelName
Nombre del canal (20 bytes rellenados a la derecha con blancos).

ChannelType
Código que define el tipo de canal.

Identificador de usuario MCA
Este grupo de tres campos se inicializa al valor del campo MCAUSER especificado en la definición del canal. Cualquier identificador de usuario especificado por la salida de seguridad en estos campos se utiliza en el control de acceso (no es aplicable a canales SDR, SVR, CLNTCONN ni CLUSSDR).

MCAUserIdentifier
Primeros 12 bytes de identificador rellenados a la derecha con blancos.

LongMCAUserIdPtr
Puntero a un búfer que contiene el identificador de longitud completa (sin garantía de terminación en nulo); tiene prioridad sobre MCAUserIdentifier.

LongMCAUserIdLength
Longitud de la cadena a la que apunta LongMCAUserIdPtr; tiene que definirse si se define LongMCAUserIdPtr.

Identificador de usuario remoto
Solo se aplica a pares de canal CLNTCONN/SVRCONN. Si no se ha definido ninguna salida de seguridad CLNTCONN, estos tres campos son inicializados por el MCA del cliente; por tanto, podrían contener un identificador de usuario del entorno del cliente que una salida de seguridad SVRCONN puede utilizar en la autenticación y al especificar el identificador de usuario de MCA. Si se define una salida de seguridad CLNTCONN, estos campos no se inicializan y los puede establecer la salida de seguridad CLNTCONN o se pueden utilizar mensajes de seguridad para pasar un identificador de usuario del cliente al servidor.

RemoteUserIdentifier
Primeros 12 bytes de identificador rellenados a la derecha con blancos.

LongRemoteUserIdPtr
Puntero a un búfer que contiene el identificador de longitud completa (sin garantía de terminación en nulo); tiene prioridad sobre RemoteUserIdentifier.

LongRemoteUserIdLength
Longitud de la cadena a la que apunta LongRemoteUserIdPtr; tiene que definirse si se define LongRemoteUserIdPtr.

PMQLONG pDataLength
entrada/salida

Puntero a MQLONG. Contiene la longitud de cualquier salida de seguridad contenida en AgentBuffer al invocarse la salida de seguridad. Tiene que ser establecido por una salida de seguridad a la longitud de cualquier mensaje que se esté enviando en AgentBuffer o ExitBuffer.

PMQLONG pAgentBufferLength
entrada

Puntero a MQLONG. Es la longitud de los datos contenidos en AgentBuffer cuando se invoca la salida de seguridad.

PMQVOID pAgentBuffer
entrada/salida

Al invocarse la salida de seguridad, este puntero apunta a cualquier mensaje enviado desde la salida asociada. Si ExitResponse2 en la estructura MQCXP tiene establecido el distintivo MQXR2_USE_AGENT_BUFFER (valor predeterminado) una salida de seguridad tiene que establecer este parámetro para que apunte a los datos de mensaje que se envíen.

PMQLONG pExitBufferLength
entrada/salida

Puntero a MQLONG. Este parámetro se inicializa a 0 en la primera invocación de una salida de seguridad y el valor devuelto se mantiene entre llamadas a la salida de seguridad durante un intercambio de seguridad.

PMQPTR pExitBufferAddr
entrada/salida

Este parámetro se inicializa a un puntero nulo en la primera invocación de una salida de seguridad y el valor devuelto se mantiene entre llamadas a la salida de seguridad durante un intercambio de seguridad. Si el distintivoMQXR2_USE_EXIT_BUFFER se establece a ExitResponse2 en la respuesta MQCXP, una salida de seguridad tiene que establecer este parámetro para que apunte a cualquier dato de mensaje que se envíe.

Diferencias de comportamiento entre las salidas de seguridad definidas en los pares de canales CLNTCONN/SVRCONN y otros pares de canales
Las salidas de seguridad se pueden definir en todos los tipos de canales. No obstante, el comportamiento de las salidas de seguridad definidas en los pares de canales CLNTCONN/SVRCONN es ligeramente diferente de las salidas de seguridad definidas en otros pares de canales.

Una salida de seguridad de un canal CLNTCONN puede establecer el identificador de usuario remoto en la definición de canal para procesar una salida SVRCONN asociada o para la autorización OAM si no se ha definido una salida de seguridad SVRCONN y no se ha establecido el campo MCAUSER de SVRCONN.

Si no se ha definido ninguna salida de seguridad CLNTCONN, el cliente de MCA establece el identificador de usuario remoto de la definición de canal en un identificador de usuario del entorno de cliente, el cual puede estar en blanco.

Un intercambio de seguridad entre las salidas de seguridad definidas en el par de canales CLNTCONN y SVRCONN se completa correctamente cuando la salida de seguridad SVRCONN devuelve una ExitResponse de MQXCC_OK. Un intercambio de seguridad entre otros pares de canales se completa correctamente cuando la salida de seguridad que ha iniciado el intercambio devuelve una ExitResponse de MQXCC_OK.

No obstante, se puede utilizar el código MQXCC_SEND_AND_REQUEST_SEC_MSG de ExitResponse para forzar la continuación del intercambio de seguridad: Si una salida de seguridad de CLNTCONN o SVRCONN devuelve una ExitResponse de MQXCC_SEND_AND_REQUEST_SEC_MSG, la salida asociada debe responder enviando un mensaje de seguridad (no MQXCC_OK o una respuesta nula) o finalizará el canal. Para las salidas de seguridad definidas en otros tipos de canal, se devuelve una ExitResponse deMQXCC_OK como respuesta a un MQXCC_SEND_AND_REQUEST_SEC_MSG de la salida de seguridad asociada, para que continúe el intercambio de seguridad como si no se hubiera devuelto una respuesta nula y no finalice el canal.

Salida de seguridad SSPI
IBM MQ for Windows ofrece una salida de seguridad que proporciona autenticación para canales IBM MQ mediante la interfaz de programación de servicios de seguridad (SSPI). La SSPI proporciona servicios de seguridad integrada de Windows.

Esta salida de seguridad es tanto para el cliente IBM MQ como para el servidor IBM MQ.

Los paquetes de seguridad se cargan desde security.dll o secur32.dll. Estas DLL se suministran con el sistema operativo.

La autenticación de una vía se proporciona en Windows, utilizando los servicios de autenticación de NTLM. La autenticación bidireccional se proporciona en Windows 2000, utilizando los servicios de autenticación de Kerberos.

El programa de salida de seguridad se proporciona en formato fuente y de objeto. Puede utilizar el código de objeto tal como está, o puede utilizar el código fuente como punto de partida para crear sus propios programas de salida de usuario. Para obtener más información sobre el uso del objeto o código fuente de la salida de seguridad SSPI, consulte "Utilización de la salida de seguridad SSPI en Windows" en la página 1164

Programas de salida de envío y recepción de canal
Puede utilizar las salidas de envío y recepción para realizar tareas como la compresión y la descompresión de datos. Se puede especificar una lista de programas de salida de envío y recepción para que ejecuten en secuencia.

Los programas de salida de envío y recepción de canal se llaman en los siguientes puntos del ciclo de proceso de un MCA:

- Los programas de salida de envío y recepción se llaman para la inicialización en la inicialización de MCA y para la terminación en la terminación de MCA.
- El programa de salida de envío se invoca en uno u otro extremo del canal, dependiendo de cuál sea el extremo en el que envía una transmisión de una transferencia de mensaje, inmediatamente antes de que la transmisión se envíe por el enlace. La nota 4 explica por qué las salidas están disponibles en ambas direcciones incluso aunque los canales de mensajes envíen mensajes solo en una dirección.
- El programa de salida de recepción se invoca en uno u otro extremo del canal, dependiendo de cuál sea el extremo en el que se recibe una transmisión de una transferencia de mensaje, inmediatamente después de que la transmisión se obtenga del enlace. La nota 4 explica por qué las salidas están disponibles en ambas direcciones incluso aunque los canales de mensajes envíen mensajes solo en una dirección.

Pueden existir muchas transmisiones para una transferencia de mensaje, y podrían haber muchas iteraciones de los programas de salida de envío y recepción antes de que un mensaje alcance la salida de mensaje en el extremo de recepción.

A los programas de salida de envío y recepción de canal se les pasa un almacenamiento intermedio de agente que contiene los datos que se envían o reciben del enlace de comunicaciones. Para los programas de salida de envío, se reservan los primeros 8 bytes del almacenamiento intermedio para su uso por parte del MCA (agente de canal de mensajes), y no se deben cambiar. Si el programa devuelve un almacenamiento intermedio distinto, estos primeros 8 bytes deben existir en el nuevo almacenamiento intermedio. El formato de los datos presentados a los programas de salida no está definido.

Los programas de salida de envío y recepción deben devolver un código de respuesta bueno. Cualquier otra respuesta provoca una terminación anómala del MCA (abend).

Nota: No emita una llamada MQGET, MQPUT o MQPUT1 dentro de un punto de sincronización desde una salida de envío o recepción.

Nota:

1. Las salidas de emisión y recepción normalmente funcionan en pares. Por ejemplo, es posible que una salida de envío comprima los datos y una salida de recepción los descomprima, o que una salida de envío cifre los datos y una salida de recepción los descifre. Al definir los canales adecuados, asegúrese de que se especifican programas de salida compatibles para ambos extremos del canal.
2. Si se activa la compresión para el canal, se pasan datos comprimidos a las salidas.
3. Es posible que se llamen salidas de envío y recepción de canal para segmentos de mensajes que no sean de datos de aplicación, por ejemplo, mensajes de estado. No se llaman durante el dialogo de inicio ni la fase de comprobación de seguridad.
4. Aunque los canales de mensajes envían mensajes solo en una dirección, los datos de control de canal como los latidos y la finalización del proceso por lotes, fluyen en ambas direcciones, y estas salidas están disponibles también en ambas direcciones. No obstante, algunos de los flujos de iniciales del inicio de canal están exentos del proceso por parte de cualquiera de las salidas.
5. Hay circunstancias en las que las salidas de envío y recepción se pueden invocar fuera de secuencia; por ejemplo, si está ejecutando una serie de programas de salida o si también está ejecutando salidas de seguridad. En este caso, cuando se llama por primera vez la salida de recepción para procesar datos, es posible que reciba datos que no han pasado por la correspondiente salida de envío. Si la salida de recepción acaba de realizar la operación, por ejemplo, la descompresión, sin comprobar primero que fuera necesaria, los resultados podrían ser inesperados.

   Necesita codificar las salidas de envío y recepción de manera que la salida de recepción pueda comprobar que los datos que recibe han sido procesados por la salida de envío correspondiente. El método recomendado para hacer esto es codificar los programas de salida de manera que:

   - La salida de envío establezca el valor del noveno byte de datos en 0 y desplace todos los datos 1 byte, antes de realizar la operación. (Los primeros 8 bytes se reservan para su uso por parte del MCA).
   - Si la salida de recepción recibe datos que tienen un 0 en el byte 9, sabe que los datos proceden de la salida de envío. Elimina el 0, realiza la operación complementaria y vuelve a desplazar los datos resultantes 1 byte.
   - Si la salida de recepción recibe datos que tienen algo distinto a 0 en el byte 9, presupone que la salida de envío no se ha ejecutado y vuelve a enviar los datos al interlocutor sin modificar.

   Al utilizar salidas de seguridad, si la salida de seguridad finaliza el canal, es posible que se llame una salida de envío sin la salida de recepción correspondiente. Una manera de evitar este problema es codificar la salida de seguridad para establecer un distintivo, en MQCD.SecurityUserData o MQCD.SendUserData, por ejemplo, cuando la salida decida finalizar el canal. A continuación, la salida de envío necesita comprobar este campo y procesar los datos únicamente si no se ha establecido el distintivo. Esta comprobación evita que la salida de envío modifique los datos innecesariamente, evitando así los errores de conversión que pudieran ocurrir si la salida de seguridad recibe datos modificados.

Programas de salida de envío de canal: reserva de espacio
Se pueden utilizar salidas de envío y recepción para transformar los datos antes de su transmisión. Los programas de salida de envío de canal pueden añadir sus propios datos sobre la transformación reservando espacio en el búfer de transmisión.

El programa de salida de recepción procesa estos datos y los elimina del búfer. Por ejemplo, puede que desee cifrar los datos y añadir una clave de seguridad para el descifrado.

Cómo reservar espacio y usarlo

Cuando se llama al programa de salida de emisión para la inicialización, establezca el campo *ExitSpace* de MQXCP en el número de bytes que se van a reservar. Consulte MQCXP para obtener detalles.

ExitSpace sólo se puede establecer durante la inicialización, es decir, cuando *ExitReason* tiene el valor MQXR_INIT. Cuando la salida de envío se invoca inmediatamente antes de la transmisión, con *ExitReason* establecido a MQXR_XMIT, se reservan *ExitSpace* bytes en el búfer de transmisión. *ExitSpace* no está soportado en z/OS.

La salida de envío no tiene por qué utilizar todo el espacio reservado. Puede utilizar menos de *ExitSpace* bytes o, si el almacenamiento intermedio de transmisión no está lleno, la salida puede utilizar más de la cantidad reservada. Al establecer el valor de *ExitSpace*, debe dejar al menos 1 KB para los datos de mensaje en el almacenamiento intermedio de transmisión. El rendimiento del canal se puede ver afectado si se utiliza el espacio reservado para grandes cantidades de datos.

El almacenamiento intermedio de transmisión es normalmente de 32KB de longitud. Sin embargo, si el canal utiliza TLS, el tamaño del búfer de transmisión se reduce a 15.352 bytes para que encaje en la longitud máxima de registro definida en la RFC 6101 y la familia de estándares TLS relacionada. Se reservan 1024 bytes adicionales para ser utilizados por IBM MQ, por lo que el espacio máximo de almacenamiento intermedio de transmisión utilizable por las salidas de envío es 14.328 bytes.

¿Qué ocurre en el extremo receptor del canal?

Los programas de salida de recepción de canal tienen que estar configurados para ser compatibles con las correspondientes salidas de envío. Las salidas de recepción tienen que conocer el número de bytes del espacio reservado y tienen que eliminar los datos de dicho espacio.

Múltiples salidas de envío

Se puede especificar una lista de programas de salida de envío y recepción para que ejecuten en secuencia. IBM MQ mantiene un total para el espacio reservado por todas las salidas de envío. Este espacio total tiene que dejar al menos 1 KB para los datos de mensaje en el búfer de transmisión.

En el ejemplo siguiente se muestra cómo se asigna el espacio a tres salidas de envío, llamadas secuencialmente:

1. Al ser invocada para su inicialización:
   - Las salida de envío A reserva 1 KB.
   - Las salida de envío B reserva 2 KB.
   - Las salida de envío C reserva 3 KB.
2. El tamaño máximo de transmisión es de 32 KB y los datos de usuario tienen una longitud de KB.
3. Se llama a la salida A con 5 KB de datos; están disponibles hasta 27 KB, porque se reservan 5 KB para las salidas B y C. La salida A añade 1 KB, el importe que ha reservado.
4. Se llama a la salida B con 6 KB de datos; están disponibles hasta 29 KB, porque se reservan 3 KB para la salida C. La salida B añade 1 KB, menos de los 2 KB que ha reservado.
5. Se llama a la salida C con 7 KB de datos; hay un máximo de 32 KB disponibles. La salida C añade 10K, más de los 3 KB que reservó. Esta cantidad es válida, porque la cantidad total de datos, 17 KB, está por debajo del máximo de 32 KB.

El tamaño máximo del búfer de transmisión de un canal que use TLS es de 15.352 bytes, no de 32 Kb. Esto se debe a que los segmentos de transmisión de socket seguros subyacentes están limitados a 16 Kb y parte del espacio es necesario para las sobrecargas de registro de TLS. Se reservan 1024 bytes adicionales para ser utilizados por IBM MQ, por lo que el espacio máximo de almacenamiento intermedio de transmisión utilizable por las salidas de envío es 14.328 bytes.

Programas de salida de mensajes de canal
Puede utilizar la salida de mensajes de canal para realizar tareas como, por ejemplo, el cifrado en el enlace, la validación o la sustitución de los ID de usuario de entrada, la conversión de datos de mensajes, el registro por diario (journaling) y el manejo de mensajes de referencia. Puede especificar una lista de programas de salida de mensajes para que se ejecuten sucesivamente.

Los programas de salida de mensajes de canal se llaman en los lugares siguientes en el ciclo de proceso del MCA:

- Durante el inicio y la finalización del MCA.
- Inmediatamente después de que un MCA emisor haya emitido una llamada MQGET
- Antes de que el MCA receptor emita una llamada MQPUT

La salida de mensaje se pasa a un almacenamiento intermedio de agente que contiene la cabecera de cola de transmisión MQXQH y el texto del mensaje de aplicación tal cual se recupera de la cola. El formato de MQXQH se proporciona en MQXQH - Transmission-queue header.

Si utiliza mensajes de referencia (es decir, mensajes que sólo contienen una cabecera que apunta a algún otro objeto que se va a enviar), la salida de mensaje reconoce la cabecera, MQRMH. Identifica el objeto, lo recupera de la forma que sea apropiada la añade a la cabecera, y la pasa al MCA para su transmisión al MCA receptor. En el MCA receptor, otra salida de mensaje reconoce que este mensaje es un mensaje de referencia, extrae el objeto y pasa la cabecera a la cola de destino. Consulte “Mensajes de referencia” en la página 816 y “Ejecución de los ejemplos de mensajes de referencia” en la página 1135 para obtener más información sobre los mensajes de referencia y algunas salidas de mensajes de ejemplo que los gestionan.

Las salidas de mensajes pueden devolver las respuestas siguientes:

- Envíe el mensaje (salida GET). Es posible que el mensaje haya sido modificado por la salida. (Esto devuelve MQXCC_OK.)
- Ponga el mensaje en la cola (salida PUT). Es posible que el mensaje haya sido modificado por la salida. (Esto devuelve MQXCC_OK.)
- No procesar el mensaje. El mensaje se coloca en la cola de mensajes no entregados (cola de mensajes no entregados) por el MCA.
- Cierre el canal.
- Código de retorno incorrecto, que hace que el MCA termine anormalmente.

Nota:

1. Las salidas de mensajes se invocan una vez para cada mensaje completo transferido, incluso cuando el mensaje se divide en partes.
2. Linux AIX Si proporciona una salida de mensajes en AIX o Linux, la conversión automática de los ID de usuarios a minúsculas (descrita aquí) no se produce.
3. Una salida se ejecuta en la misma hebra que el propio MCA. También se ejecuta dentro de la misma unidad de trabajo (UOW) que el MCA porque utiliza el mismo manejador de conexión. Por lo tanto, cualquier llamada realizada bajo el punto de sincronismo se confirma o restituye por el canal al final del lote. Por ejemplo, un programa de salida de mensaje de canal puede enviar mensajes de notificación a otro y estos mensajes sólo se confirman en la cola cuando se confirma el lote que contiene el mensaje original.

   Por lo tanto, puede emitir llamadas MQI de punto de sincronismo desde un programa de salida de mensajes de canal.

Conversión de mensajes fuera de una salida de mensaje
Antes de invocar una salida de mensaje, el MCA receptor realiza algunas conversiones en el mensaje. En este tema se describen los algoritmos que se utilizan para realizar las conversiones.

Qué cabeceras se procesan

Se ejecuta una rutina de conversión en el MCA del receptor antes de invocarse la salida de mensaje. La rutina de conversión empieza por la cabecera MQXQH al principio del mensaje. A continuación, la rutina de conversión procesa las cabeceras encadenadas que siguen a MQXQH, realizando la conversión cuando es necesario. Las cabeceras encadenadas se pueden extender más allá del desplazamiento contenido en

el parámetro HeaderLength de los datos MQCXP que se pasan a la salida de mensaje del destinatario. Las cabeceras siguientes se convierten in situ:

- MQXQH (nombre de formato " MQXMIT ")
- MQMD (esta cabecera forma parte de MQXQH y no tiene nombre de formato)
- MQMDE (nombre de formato " MQHMDE ")
- MQDH (nombre de formato " MQHDIST ")
- MQWIH (nombre de formato " MQHWIH ")

Las cabeceras siguientes no se convierten, sino que se saltan a medida que el MCA procesa las cabeceras encadenadas:

- MQDLH (nombre de formato " MQDEAD ")
- cualquier cabecera con nombre de formato que empiece por los tres caracteres 'MQH' (por ejemplo, " MQHRF ")

Cómo se procesan las cabeceras

El parámetro Format de cada cabecera de IBM MQ es leído por MCA. El parámetro Format tiene 8 bytes dentro de la cabecera, que son de 8 caracteres de un byte que contienen un nombre.

A continuación, el MCA interpreta los datos que siguen a cada cabecera son del tipo indicado. Si Format es el nombre de un tipo de cabecera elegible para la conversión de IBM MQ, se convierte. Si se trata de otro nombre que indica que no son datos de MQ (por ejemplo, MQFMT_NONE o MQFMT_STRING), el MCA deja de procesar las cabeceras.

¿Qué es el HeaderLength de MQCXP?

El parámetro HeaderLength en los datos MQCXP proporcionados a una salida de mensaje es la longitud total de las cabeceras MQXQH (lo que incluye el MQMD), MQMDE y MQDH al inicio del mensaje. Estas cabeceras se encadenan usando los nombres y longitudes de 'Format'.

MQWIH

Las cabeceras encadenadas se pueden extender más allá de HeaderLength en el área de datos de usuario. La cabecera MQWIH, si está presente, es una de las cabeceras que aparecen más allá de HeaderLength.

Si hay una cabecera MQWIH en las cabeceras encadenadas, se convierte in situ antes de que se invoque la salida de mensaje del destinatario.

Programa de salida de reintento de mensaje
La salida de reintento de mensaje de canal se invoca cuando un intento de abrir la cola de destino no se realiza correctamente. Puede utilizar la salida para determinar en qué circunstancias se ha de reintentar, cuántas se ha de reintentar y con qué frecuencia.

Esta salida también se invoca en el extremo receptor del canal durante la iniciación y terminación de MCA.

La salida de reintento de mensaje se pasa a un almacenamiento de agente que contiene la cabecera la cola de transmisión, MQXQH, y el texto del mensaje de aplicación como se ha recuperado de la cola. El formato de MQXQH se proporciona en la sección Visión general de MQXQH.

La salida se invoca para todos los códigos de razón. La salida determina para qué códigos de razón MCA desea que se realice el reintento, el número de veces y los intervalos. El valor del número de reintentos de mensaje que se ha establecido cuando se ha definido el canal se pasa a la salida en el MQCD, pero la salida puede omitir este valor.

Cada vez que se invoca la salida, MCA incrementa el campo MsgRetryCount de MQCXP, y la salida devuelve MQXCC_OK con el tiempo de espera que indica el campo MsgRetryInterval de MQCXP o MQXCC_SUPPRESS_FUNCTION. Los reintentos continúan de forma indefinida hasta que la salida

devuelve MQXCC_SUPPRESS_FUNCTION en el campo ExitResponse de MQCXP. Consulte la sección MQCXP para obtener información acerca de la acción que ha de realizar MCA para estos códigos de terminación.

Si todos los reintentos no se ejecutan correctamente, el mensaje se coloca en la cola de mensajes no entregados. Donde no haya ninguna cola de mensajes no entregados disponible, se detiene el canal.

Si no define una salida de reintento de mensaje para un canal y se produce un error, es probable que éste sea temporal, por ejemplo MQRC_Q_FULL, ya que MCA utiliza el número de reintentos de mensajes y los intervalos de reintentos de mensaje establecidos en la definición del canal. Si el error tiene una naturaleza más permanente y no ha definido un programa de salida para manejarlo, el mensaje se coloca en la cola de mensajes no entregados.

Programa de salida de definición automática de canal
La salida de definición automática de canal se puede utilizar cuando se recibe una solicitud para iniciar un canal receptor o de conexión con el servidor, pero no existe ninguna definición para ese canal (no para IBM MQ for z/OS). También puede llamarse en todas las plataformas para canales de clúster emisor y de clúster receptor para la modificación para una instancia del canal.

La salida de definición automática de canal se puede llamar en todas las plataformas excepto en z/OS cuando se recibe una petición para iniciar un canal de receptor o de conexión de servidor, pero no existe ninguna definición de canal. Puede utilizarla para modificar la definición suministrada predeterminada para un canal receptor o de conexión con el servidor definido, SYSTEM.AUTO.RECEIVER o SYSTEM.AUTO.SVRCON. Consulte Preparación de canales para obtener una descripción de cómo se pueden crear automáticamente las definiciones de canal.

La salida de definición automática de canal también se puede llamar cuando se recibe una solicitud para iniciar un canal de clúster emisor. Se puede llamar para canales de clúster emisor o de clúster receptor para permitir la modificación de la definición para esta instancia de canal. En este caso, la salida también se aplica a IBM MQ for z/OS. Un uso común de salida de definición automática del canal es cambiar los nombres de las salidas de mensajes (MSGEXIT, RCVEXIT, SCYEXIT y SENDEXIT) debido a que los nombres de salida tienen formatos distintos en plataformas distintas. Si no se especifica ninguna salida de definición automática de canal, el comportamiento predeterminado en z/OS es examinar un nombre de salida distribuida con el formato *[path]/libraryname(function)* y tomar hasta ocho caracteres de función, si está presente, o nombre de biblioteca. En z/OS, un programa de salida de definición automática de canal debe modificar los campos a los que hacen referencia MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr y ReceiveUserDataPtr, en lugar de los propios campos de MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit y ReceiveUserData.

Para obtener más información, consulte Cómo trabajar con canales definidos automáticamente.

Al igual que otras salidas de canal, la lista de parámetros es:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` se describen en MQCXP. `ChannelDefinition` se describe en MQCD.

MQCD contiene los valores que se utilizan en la definición de canal predeterminada si la salida no los altera. La salida puede modificar solamente un subconjunto de los campos; consulte MQ_CHANNEL_AUTO_DEF_EXIT. No obstante, intentar cambiar otros campos no causa un error.

La salida de definición automática de canal devuelve una respuesta de MQXCC_OK o MQXCC_SUPPRESS_FUNCTION. Si no se devuelve ninguna de estas respuestas, el MCA continúa procesándose como si fuera devuelto MQXCC_SUPPRESS_FUNCTION. Es decir, se abandona la definición automática, no se crea ninguna definición de canal nueva y el canal no se puede iniciar.

ALW *Compilación de programas de salida de canal en sistemas AIX, Linux, and Windows*

Utilice los ejemplos siguientes como ayuda para compilar programas de salida de canal para sistemas AIX, Linux, and Windows .

Windows

Windows

El mandato de compilador y enlazador para los programas de salida de canal en Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

Sistemas AIX and Linux

Linux AIX

En estos ejemplos `exit` es el nombre de la biblioteca y `ChannelExit` es el nombre de la función. En AIX, el nombre del archivo de exportación es `exit.exp`. La definición de canal utiliza estos nombres para hacer referencia al programa de salida con el formato descrito en la sección MQCD, definición de canal. Consulte también el parámetro MSGEXIT del mandato DEFINE CHANNEL.

AIX Mandatos de compilador y enlazador de ejemplo para salidas de canal en AIX:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Mandatos de compilador y enlazador de ejemplo para salidas de canal en Linux, cuando el gestor de colas es de 32 bits:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Mandatos de compilador y enlazador de ejemplo para salidas de canal en Linux, cuando el gestor de colas es de 64 bits:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

En el cliente, se puede utilizar una salida de 32 bits o de 64 bits. Esta salida se debe enlazar a mqic_r.

AIX En AIX, todas las funciones invocadas mediante IBM MQ se deben exportar. El siguiente es un archivo de exportación de ejemplo para este archivo make:

```
#
!channelExit
MQStart
```

Configuración de una salida de canal

Para invocar una salida de canal, hay que nombrarla en la definición de canal.

Las salidas de canal tienen que nombrarse en la definición de canal. Puede efectuar este nombrado la primera vez que defina canales o puede añadir la información posteriormente utilizando, por ejemplo, el comando MQSC "ALTER CHANNEL". También se pueden facilitar los nombres de salida al canal en la estructura de datos de canal MQCD. El formato del nombre de salida depende de la plataforma IBM MQ; consulte MQCD o Mandatos MQSC para obtener más información.

Si la definición de canal no contiene el nombre de programa de una salida de usuario, no se invocará dicha salida de usuario.

La salida de definición automática de canal es la propiedad del gestor de colas, no del canal individual. Para que se invoque esta salida, hay que nombrarla en la definición del gestor de colas. Para modificar una definición de gestor de colas, utilice el comando MQSC ALTER QMGR.

Escribir salidas de conversión de datos

Este grupo de temas contiene información sobre cómo escribir salidas de conversión de datos.

Nota: No está soportado en MQSeries para VSE/ESA.

Cuando se realiza una llamada MQPUT, la aplicación crea el descriptor de mensaje (MQMD) del mensaje. Dado que IBM MQ necesita comprender el contenido del MQMD, independientemente de la plataforma en que se ha creado, el sistema lo convierte automáticamente.

No obstante, los datos de la aplicación no se convierten automáticamente. Si se intercambian datos de caracteres entre plataformas donde los campos `CodedCharSetId` y `Encoding` difieren, por ejemplo entre ASCII y EBCDIC, la aplicación debe organizar la conversión del mensaje. La conversión de datos de la aplicación la puede realizar el propio gestor de colas o un programa de salida de usuario, denominado *salida de conversión de datos*. El gestor de colas puede realizar él mismo la conversión de datos, utilizando una de las rutinas de conversión incorporadas, si los datos de aplicación están en uno de los formatos incorporados (como por ejemplo, MQFMT_STRING). Este tema incluye información acerca del recurso de salida de conversión de datos que proporciona IBM MQ para los casos en los que los datos de la aplicación no están en un formato incluido.

El control se puede pasar a la salida de conversión de datos durante una llamada MQGET. Esto evita la conversión en distintas plataformas antes de que se llegue al destino final. Sin embargo, si el destino final es una plataforma que no soporta la conversión de datos en la MQGET, debe especificar CONVERT(YES) en el canal emisor que envía los datos a su destino final. Esto garantiza que IBM MQ convierta los datos durante la transmisión. En este caso, la salida de conversión de datos debe residir en el sistema donde está definido el canal emisor.

La aplicación emite directamente la llamada MQGET. Establezca los campos `CodedCharSetId` y `Encoding` del MQMD en el juego de caracteres y la codificación necesarios. Si la aplicación utiliza el mismo juego de caracteres y la misma codificación que el gestor de colas, establezca `CodedCharSetId` en MQCCSI_Q_MGR y `Encoding` en MQENC_NATIVE. Después de que la llamada MQGET se complete, estos campos tienen los valores apropiados para los datos de mensaje devueltos. Estos pueden diferir de los valores necesarios si la conversión no ha sido satisfactoria. La aplicación debe restablecer estos campos en los valores necesarios antes de cada llamada MQGET.

Las condiciones necesarias para invocar la salida de conversión de datos se definen para la llamada MQGET en MQGET.

Para obtener una descripción de los parámetros que se pasan a la salida de conversión de datos, y notas de uso detalladas, consulte Conversión de datos para la llamada MQ_DATA_CONV_EXIT y la estructura MQDXP.

Los programas que convierten datos de aplicación entre diferentes codificaciones de máquina y CCSID, deben ser compatibles con la interfaz de conversión de datos (DCI) de IBM MQ.

Para clientes de multidifusión, las salidas de API y las salidas de conversión de datos se deben poder ejecutar en el lado del cliente porque es posible que algunos mensajes no pasen por el gestor de colas. Las siguientes bibliotecas forman parte de los paquetes de cliente así como de los paquetes de servidor:

Tabla 143. Bibliotecas incluidas en los paquetes de cliente y servidor

Sistema operativo	**Bibliotecas**
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bits & 64 bits: libmqm.so & libmqm_r.so
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb

Invocación de la salida de conversión de datos

Una salida de conversión de datos es una salida escrita por el usuario que recibe el control durante el procesamiento de una llamada MQGET.

La salida se invoca si se cumplen las siguientes sentencias:

- La opción MQGMO_CONVERT se especifica en la llamada MQGET.
- Algunos o todos los datos de mensaje no están en el juego de caracteres solicitado o en la codificación.
- El campo *Format* de la estructura MQMD asociada al mensaje no es MQFMT_NONE.
- El parámetro *BufferLength* especificado en la llamada MQGET no es cero.
- La longitud de los datos del mensaje no es cero.
- El mensaje contiene datos que tienen un formato definido por el usuario. El formato definido por el usuario puede ocupar todo el mensaje, o ir precedido de uno o más formatos incorporados. Por ejemplo, el formato definido por el usuario puede estar precedido de un formato MQFMT_DEAD_LETTER_HEADER. La salida se invoca para convertir solo el formato definido por el usuario; el gestor de colas convierte los formatos incorporados que preceden al formato definido por el usuario.

  También se puede invocar una salida escrita por el usuario para convertir un formato incorporado, pero esto solo sucede si las rutinas de conversión incorporadas no pueden convertir el formato incorporado correctamente.

Hay algunas otras condiciones, que se describen completamente en las notas de uso de la llamada MQ_DATA_CONV_EXIT en MQ_DATA_CONV_EXIT.

Consulte MQGET para obtener los detalles de la llamada MQGET. Las salidas de conversión de datos no pueden utilizar llamadas MQI distintas de MQXCNVC.

Se carga una nueva copia de la salida cuando una aplicación intenta recuperar el primer mensaje que utiliza ese *Format* desde que la aplicación se conectó con el gestor de colas. Puede que también se cargue una copia nueva en otras ocasiones si el gestor de colas ha descartado una copia cargada previamente.

La salida de conversión de datos ejecuta en un entorno como el del programa que ha emitido la llamada MQGET. Al igual que las aplicaciones de usuario, el programa puede ser un agente de canal de mensaje (Message Channel Agent, MCA) que esté enviando mensajes a un gestor de colas de destino que no soporta conversión de mensajes. El entorno incluye un espacio de direcciones y un perfil de usuario, cuando procede. La salida no puede comprometer la integridad del gestor de colas, porque no ejecuta en el entorno del gestor de colas.

Conversión de datos en z/OS

z/OS

En z/OS, tenga en cuenta lo siguiente:

- Los programas de salida solo se pueden escribir en ensamblador.
- Los programas de salida tienen que ser reentrantes y ser capaces de ejecutar en cualquier lugar del almacenamiento.
- Al salir, los programas de salida tienen que restaurar el entorno que había al entrar y liberar cualquier almacenamiento obtenido.
- Los programas de salida no pueden ser WAIT ni emitir ESTEEs o SPIEs.
- Los programas de salida se suelen invocar como si lo hiciera z/OS LINK en:
  - Estado de programa anómalo no autorizado
  - Modalidad de control de espacio de direcciones primaria
  - Modo no entre memorias (no cross-memory).
  - Modo no de registro de acceso.

  - Modalidad de direccionamiento de 31 bits
  - Modo TCB-PRB.
- Cuando la utiliza una aplicación CICS, la salida la invoca EXEC CICS LINK, y debe ser compatible con las convenciones de programación de CICS. Los parámetros se pasan mediante punteros (direcciones) en el área de comunicación de CICS (COMMAREA).

  Aunque no se recomienda, los programas de salida de usuario también pueden utilizar llamadas de API CICS, con la precaución siguiente:

  - No emita puntos de sincronización, ya que los resultados podrían influir en las unidades de trabajo declaradas por el MCA.
  - No actualice ningún recurso controlado por un gestor de recursos distinto de IBM MQ for z/OS, incluidos los controlados por CICS Transaction Server.

  En los canales con CONVERT=YES, la salida se carga desde el conjunto de datos referenciado por la sentencia CSQXLIB DD. Las salidas proporcionadas por MQ CSQCBDCI y CSQCBDCO para el puente IBM MQ CICS están en SCSQAUTH.

Escritura de un programa de salida de conversión de datos para IBM i

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos de MQ para IBM i.

Siga estos pasos:

1. Nombre el formato del mensaje. El nombre tiene que encajar en el campo *Format* del MQMD. El nombre *Format* no debe tener espacios en blanco intercalados iniciales y los espacios en blanco finales se ignoran. El nombre del objeto no puede tener más de ocho caracteres distintos del espacio en blanco, porque *Format* solo tiene ocho caracteres de longitud. Recuerde que debe utilizar este nombre cada vez que envíe un mensaje (en nuestro ejemplo se utiliza el nombre Formato).
2. Cree una estructura para representar el mensaje. Consulte Sintaxis válida para obtener un ejemplo.
3. Ejecute esta estructura mediante el mandato CVTMQMDTA para crear un fragmento de código para la salida de conversión de datos.

   Las funciones generadas por el mandato CVTMQMDTA utilizan macros que se incluyen en el archivo QMQM/H(AMQSVMHA). Estas macros se escriben presuponiendo que todas las estructuras están empaquetadas; añádalas si no es así.
4. Realice una copia del archivo de origen de esqueleto proporcionado, QMQMSAMP/QCSRC(AMQSVFC4), y cámbiele el nombre. (En nuestro ejemplo se utiliza el nombre EXIT_MOD).
5. Busque los siguientes recuadros de comentarios en el archivo del código fuente e inserte el código tal como se describe:

   a. Cerca del final del archivo del código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Aquí, inserte el fragmento de código generado en el paso “3” en la página 1011.

   b. Por el centro del archivo de código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Esto va seguido de una llamada comentada a la función `ConverttagSTRUCT`.

   Cambie el nombre de la función al nombre de la función añadida en el paso “5.a” en la página 1011. Elimine los caracteres de comentario para activar la función. Si hay varias funciones, cree llamadas para cada una de ellas.

   c. Cerca del principio del archivo de código fuente, un recuadro de comentarios empieza por:

```
/* Insert the function prototypes for the functions produced by */
```

Aquí, inserte las sentencias de prototipo de función para las funciones añadidas en el paso “5.a” en la página 1011.

Si el mensaje contiene datos de caracteres, el código generado llama a MQXCNVC; esto se puede resolver enlazando el programa de servicio QMQM/LIBMQM.

6. Compile el módulo fuente, EXIT_MOD, según se indica a continuación:

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. Cree/enlace el programa.

   Para aplicaciones sin hebras, utilice lo siguiente:

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

   Además de crear la salida de conversión de datos para el entorno básico, se necesita otra en el entorno con hebras. Este objeto cargable debe ir seguido por _R. Utilice la biblioteca LIBMQM_R para resolver llamadas a MQXCNVC. Ambos objetos cargables son necesarios para un entorno con hebras.

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. Coloque la salida en la lista de bibliotecas del trabajo de IBM MQ. Se recomienda que, para producción, los programas de salida de conversión de datos se almacenen en QSYS.

Nota:

1. Si CVTMQMDTA utiliza estructuras empaquetadas, todas las aplicaciones de IBM MQ deben utilizar el calificador _Packed.
2. Los programas de salida de conversión de datos deben ser reentrantes.
3. MQXCNVC es la única llamada MQI que se puede emitir desde una salida de conversión de datos.
4. Compile el programa de salida con la opción de compilador de perfil de usuario establecida en *USER, de manera que la salida se ejecute con la autorización del usuario.
5. La habilitación de memoria de teraespacio es necesaria para todas las salidas de usuario con IBM MQ for IBM i; especifique TERASPACE(*YES *TSIFC) en los mandatos CRTCMOD y CRTBNDC.

Escritura de un programa de salida de conversión de datos para IBM MQ for z/OS

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos para IBM MQ for z/OS.

Siga estos pasos:

1. Utilice el esqueleto de origen suministrado CSQ4BAX9 (para los entornos no CICS) o CSQ4CAX9 (para CICS) como punto de partida.
2. Ejecute el programa de utilidad CSQUCVX.
3. Siga las instrucciones del prólogo de CSQ4BAX9 o CSQ4CAX9 para incorporar las rutinas generadas por el programa de utilidad CSQUCVX, en el orden en el que aparecen las estructuras en el mensaje que desea convertir.

4. En el programa de utilidad, se supone que las estructuras de datos no están empaquetadas, que la alineación implícita de los datos se respeta y que las estructuras empiezan en un límite de palabra completa, donde los bytes se omiten según sea necesario (como entre ID y VERSION en el ejemplo de Sintaxis válida). Si las estructuras están empaquetadas, omita las macros CMQXCALA que se generan. Por lo tanto, se recomienda declarar sus estructuras de forma que se nombren todos los campos y no se omitan los bytes; en el ejemplo de Sintaxis válida, añada un campo "MQBYTE DUMMY;" entre ID y VERSION.
5. La salida proporcionada devuelve un error si el almacenamiento intermedio de entrada es más corto que el formato de mensaje que se va a convertir. Aunque la salida convierte tantos campos completos como sea posible, el error hace que se devuelva un mensaje sin convertir a la aplicación. Si desea permitir que los almacenamientos intermedios de entrada cortos se conviertan en la medida de lo posible, incluidos los campos parciales, cambie el valor TRUNC= en la macro CSQXCDFA a YES: no se devuelve ningún error, por lo que la aplicación recibe un mensaje convertido. La aplicación debe manejar el truncamiento.
6. Añada cualquier otro código de proceso especial que necesite.
7. Renombre el programa al nombre del formato de datos.
8. Compile y edite con enlaces el programa como un programa de aplicación por lotes (a menos que vaya a utilizarse con aplicaciones CICS). Las macros en el código generado por el programa de utilidad están en la biblioteca, **thlqual**.SCSQMACS.

   Si el mensaje contiene datos de caracteres, el código generado llama a MQXCNVC. Si la salida utiliza esta llamada, edítela mediante enlaces con el programa de apéndice de salida CSQASTUB. El apéndice es independiente del lenguaje e independiente del entorno. De forma alternativa, puede cargar el apéndice dinámicamente utilizando el nombre de llamada dinámica CSQXCNVC. Consulte “Llamada dinámica al apéndice IBM MQ” en la página 1055 para obtener más información.

   Coloque el módulo editado mediante enlaces en la biblioteca de carga de aplicaciones y en un conjunto de datos al que hace referencia la sentencia CSQXLIB DD del procedimiento de tarea iniciado por el iniciador de canal.
9. Si la salida es para su uso en las aplicaciones CICS, compílela y edítela mediante enlaces como un programa de aplicación CICS, incluido CSQASTUB, si es necesario. Colóquela en la biblioteca del programa de aplicación CICS. Defina el programa en CICS de la forma habitual, especificando EXECKEY (CICS) en la definición.

Nota: Aunque las bibliotecas de tiempo de ejecución LE/370 son necesarias para ejecutar el programa de utilidad CSQUCVX (consulte el paso “2” en la página 1012), no son necesarias para la edición mediante enlaces o la ejecución de la propia salida de conversión de datos (consulte los pasos “8” en la página 1013 y “9” en la página 1013).

Consulte “Escritura de aplicaciones puente IMS” en la página 76 para obtener información sobre la conversión de datos en el puente IBM MQ - IMS.

Linux AIX *Escritura de una salida de conversión de datos para sistemas IBM MQ for AIX or Linux*

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos para sistemas IBM MQ for AIX or Linux .

Siga estos pasos:

1. Nombre el formato del mensaje. El nombre tiene que encajar en el campo *Format* del MQMD y estar en mayúsculas; por ejemplo, MYFORMAT. El nombre de *Format* no puede empezar por espacios en blanco. Los espacios en blanco finales se ignoran. El nombre del objeto no puede tener más de ocho caracteres distintos del espacio en blanco, porque *Format* solo tiene ocho caracteres de longitud. No olvide usar este nombre cada vez que envíe un mensaje.

   Si la salida de conversión de datos se utiliza en un entorno con hebras, el objeto cargable tiene que ir seguido de _r para indicar que es una versión con hebras.
2. Cree una estructura para representar el mensaje. Consulte Sintaxis válida para obtener un ejemplo.

3. Ejecute esta estructura mediante el comando `crtmqcvx` para crear un fragmento de código para la salida de conversión de datos.

   Las funciones generadas por el comando `crtmqcvx` utilizan macros que se asumen que todas las estructuras están empaquetadas; corríjalas si este no fuera el caso.
4. Copie el archivo fuente del esqueleto suministrado renombrándolo al nombre del formato de mensaje configurado en el paso "1" en la página 1013. El archivo de código fuente esqueleto y la copia son de solo lectura.

   El archivo de código fuente de esqueleto se llama amqsvfc0.c.
5. En IBM MQ for AIX, también se proporciona un archivo de exportación de esqueleto llamado amqsvfc.exp. Copie este archivo, renombrándolo a MYFORMAT.EXP.
6. El esqueleto incluye un archivo de cabecera de ejemplo, amqsvmha.h, en el directorio *MQ_INSTALLATION_PATH*/inc, donde *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el cual está instalado IBM MQ. Asegúrese de que la ruta de inclusión apunta a este directorio para seleccionar este archivo.

   El archivo amqsvmha.h contiene macros usadas por el código generado por el comando `crtmqcvx`. Si la estructura que se va a convertir contiene datos de caracteres, estas macros invocan MQXCNVC.
7. Busque los siguientes recuadros de comentarios en el archivo del código fuente e inserte el código tal como se describe:

   a. Cerca del final del archivo del código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Aquí, inserte el fragmento de código generado en el paso "3" en la página 1014.

   b. Por el centro del archivo de código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Esto va seguido de una llamada comentada a la función `ConverttagSTRUCT`.

   Cambie el nombre de la función al nombre de la función añadida en el paso "7.a" en la página 1014. Elimine los caracteres de comentario para activar la función. Si hay varias funciones, cree llamadas para cada una de ellas.

   c. Cerca del principio del archivo de código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Aquí, inserte las sentencias de prototipo de función para las funciones añadidas en el paso "3" en la página 1014.
8. Compile la salida como una biblioteca compartida utilizando MQStart como punto de entrada. Para hacerlo, consulte "Compilación de salidas de conversión de datos en sistemas AIX and Linux" en la página 1015.
9. Coloque la salida en el directorio de salida. El directorio de salida predeterminado es `/var/mqm/exits` en sistemas de 32 bits y `/var/mqm/exits64` en sistemas de 64 bits. Puede cambiar estos directorios en los archivos qm.ini o mqclient.ini. Esta ruta se puede definir para cada gestor de colas y la salida solo se buscará en esa ruta o rutas.

Nota:

1. Si `crtmqcvx` utiliza estructuras empaquetadas, todas las aplicaciones IBM MQ se deben compilar de esta forma.
2. Los programas de salida de conversión de datos deben ser reentrantes.
3. MQXCNVC es la única llamada MQI que se puede emitir desde una salida de conversión de datos.

Linux AIX *Compilación de salidas de conversión de datos en sistemas AIX and Linux*

Ejemplos de cómo compilar una salida de conversión de datos en sistemas AIX and Linux.

En todas las plataformas, el punto de entrada al módulo es MQStart.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

AIX

AIX

Compile el código fuente de la salida emitiendo uno de los comandos siguientes:

Aplicaciones de 32 bits

Sin hebras

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

Compile el código fuente de la salida emitiendo uno de los comandos siguientes:

Aplicaciones de 31 bits

Sin hebras

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 32 bits

Sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Windows *Desarrollo de una salida de conversión de datos para IBM MQ for Windows*

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos para IBM MQ for Windows.

Siga estos pasos:

1. Nombre el formato del mensaje. El nombre tiene que encajar en el campo *Format* del MQMD. El nombre de *Format* no puede empezar por espacios en blanco. Los espacios en blanco finales se ignoran. El nombre del objeto no puede tener más de ocho caracteres distintos del espacio en blanco, porque *Format* solo tiene ocho caracteres de longitud.

   También se proporciona un archivo .DEF denominado amqsvfcn.def en el directorio de ejemplos, *MQ_INSTALLATION_PATH*\Tools\C\Samples. *MQ_INSTALLATION_PATH* es el directorio donde está instalado IBM MQ. Haga una copia de este archivo y renómbrela; por ejemplo, a MYFORMAT.DEF. Asegúrese de que coincidan el nombre de la DLL que se está creando y el nombre especificado en MYFORMAT.DEF. Sobrescriba el nombre FORMAT1 en MYFORMAT.DEF con el nuevo nombre de formato.

   No olvide usar este nombre cada vez que envíe un mensaje.
2. Cree una estructura para representar el mensaje. Consulte Sintaxis válida para obtener un ejemplo.
3. Ejecute esta estructura mediante el comando `crtmqcvx` para crear un fragmento de código para la salida de conversión de datos.

   Las funciones generadas por el comando CRTMQCVX utilizan macros que se escriben presuponiendo que todas las estructuras están empaquetadas; corríjalas si este no fuera el caso.
4. Copie el archivo fuente del esqueleto suministrado, amqsvfc0.c, renombrándolo al nombre del formato de mensaje configurado en el paso “1” en la página 1016.

   amqsvfc0.c se encuentra en *MQ_INSTALLATION_PATH*\Tools\C\Samples , donde *MQ_INSTALLATION_PATH* es el directorio donde está instalado IBM MQ . (El directorio de instalación predeterminado es C:\Archivos de programa\IBM\MQ.)

El esqueleto incluye un archivo de cabecera de ejemplo amqsvmha.h en el directorio *MQ_INSTALLATION_PATH*\Tools\C\include . Asegúrese de que la ruta de inclusión apunta a este directorio para seleccionar este archivo.

El archivo amqsvmha.h contiene macros usadas por el código generado por el comando CRTMQCVX. Si la estructura que se va a convertir contiene datos de caracteres, estas macros invocan MQXCNVC.

5. Busque los siguientes recuadros de comentarios en el archivo del código fuente e inserte el código tal como se describe:

   a. Cerca del final del archivo del código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Aquí, inserte el fragmento de código generado en el paso "3" en la página 1016.

   b. Por el centro del archivo de código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Esto va seguido de una llamada comentada a la función ConverttagSTRUCT.

   Cambie el nombre de la función al nombre de la función añadida en el paso "5.a" en la página 1017. Elimine los caracteres de comentario para activar la función. Si hay varias funciones, cree llamadas para cada una de ellas.

   c. Cerca del principio del archivo de código fuente, un recuadro de comentarios empieza por:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Aquí, inserte las sentencias de prototipo de función para las funciones añadidas en el paso "3" en la página 1016.

6. Cree el siguiente archivo de comandos:

   ```
   cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
   MYFORMAT.C


   MYFORMAT.DEF
   ```

   donde *MQ_INSTALLATION_PATH* es el directorio en el que IBM MQ está instalado.

7. Emita el archivo de comandos para compilar la salida como un archivo DLL.
8. Coloque la salida en el subdirectorio de salida debajo del directorio de datos de IBM MQ. El directorio predeterminado para instalar las salidas en sistemas de 32 bits es *MQ_DATA_PATH*\Exits y en sistemas de 64 bits *MQ_DATA_PATH*\Exits64

   La ruta que se usa para buscar las salidas de conversión de datos se facilita en el registro. La carpeta de registro es:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
   MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
   ```

   y la clave de registro es: ExitsDefaultPath. Esta ruta se puede definir para cada gestor de colas y la salida solo se buscará en esa ruta o rutas.

Nota:

1. Si CRTMQCVX utiliza estructuras empaquetas, todas las aplicaciones IBM MQ se deben compilar de la misma forma.
2. Los programas de salida de conversión de datos deben ser reentrantes.

3. MQXCNVC es la única llamada MQI que se puede emitir desde una salida de conversión de datos.

Windows *Archivos de carga de salida y conmutación en sistemas operativos Windows*

Los procesos del gestor de colas IBM WebSphere MQ for Windows 7.5 son de 32-bits. Como resultado, cuando se utilizan aplicaciones de 64 bits, algunos tipos de archivos de carga de salida y de conmutación XA también requieren una versión de 32 bits disponible para que la use el gestor de colas. Si se necesita la versión de 32 bits de del archivo de carga de salida o de conmutación XA y no está disponible, el correspondiente comando o llamada de API fallará.

Se da soporte a dos atributos en `qm.ini file` para *ExitPath*. Estos son `ExitsDefaultPath=` *MQ_INSTALLATION_PATH*`\exits` y `ExitsDefaultPath64=` *MQ_INSTALLATION_PATH*`\exits64`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ. El uso de estos atributos garantiza que se pueda encontrar la biblioteca adecuada. Si se usa una salida en un clúster IBM MQ, esto también garantiza que se encuentre la correspondiente biblioteca en el sistema remoto.

En la tabla siguiente se listan los distintos tipos de archivo de carga de salida y de conmutación y se indica si se necesita la versión de 32 bits o la de 64, o ambas, en función de si se están utilizando aplicaciones de 32 o de 64 bits:

Tipos de archivo	Aplicaciones de 32 bits	Aplicaciones de 64 bits
salida de API	32 bits y 64 bits	64 bits
Salida de conversión de datos	32 bits	64 bits
Salidas de canal servidor (todos los tipos)	64 bits	64 bits
Salidas de canal cliente (todos los tipos)	32 bits	64 bits
Salida de servicio instalable	64 bits	64 bits
Salida de WLM de clúster	64 bits	64 bits
Salida de direccionamiento Pub/Sub	64 bits	64 bits
Archivos de carga de conmutación de base de datos	32 bits y 64 bits	64 bits
Bibliotecas XA de gestor de transacciones externo	32 bits	64 bits
Salida de preconexión	32 bits	64 bits

Referencia a las definiciones de conexión utilizando una salida de preconexión desde un depósito

Los IBM MQ MQI clients se pueden configurar para que busquen en un repositorio y obtengan las definiciones de conexiones utilizan una biblioteca de salidas de preconexión.

Introducción

Una aplicación de cliente se puede conectar a un gestor de colas utilizando tablas de definiciones de canal de cliente (CCDT). Generalmente, el archivo CCDT se encuentra en un servidor de archivos de red central y tiene clientes que hacen referencia al mismo. Dado que es difícil gestionar y administrar varias aplicaciones de cliente que hacen referencia al archivo CCDT, un método flexible es almacenar las definiciones de cliente en un repositorio global, tal como un directorio LDAP, un registro y repositorio de WebSphere o cualquier otro repositorio. Almacenar las definiciones de conexión de cliente en un

repositorio facilita la gestión de definiciones de conexión de cliente y las aplicaciones pueden acceder a las definiciones de conexión de cliente correctas y las más actuales.

Durante la ejecución de la llamada MQCONN/X, el IBM MQ MQI client carga una biblioteca de salida de preconexión especificada por una aplicación e invoca una función de salida para recuperar definiciones de conexión. Las definiciones de conexión recuperadas se utilizan a continuación para establecer conexión con un gestor de colas. Los detalles de la biblioteca de salida y la función que se va a invocar se especifican en el archivo de configuración mqclient.ini.

Sintaxis

`void MQ_PRECONNECT_EXIT` (*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*);

Parámetros

pExitParms
Tipo: PMQNXP entrada/salida

La estructura del parámetro de salida **`PreConnection`**.

El invocador de la salida asigna y mantiene dicha estructura.

pQMgrName
Tipo: PMQCHAR entrada/salida

Nombre del gestor de colas.

En la entrada, este parámetro es la serie de filtro suministrada a la llamada de la API MQCONN a través del parámetro **`QMgrName`**. Este campo se puede estar en blanco, ser explícito o contener determinados caracteres de comodín. El campo lo modifica la salida. El parámetro es NULL cuando se invoca la salida con MQXR_TERM.

ppConnectOpts
Tipo: ppConnectOpts entrada/salida

Opciones que controlan la acción de MQCONNX.

Este es un puntero a una estructura de opciones de conexión MQCNO que controla la acción de la llamada a la API MQCONN. El parámetro es NULL cuando se invoca la salida con MQXR_TERM. El cliente MQI siempre proporciona una estructura MQCNO a la salida, aunque la aplicación no la haya proporcionado originalmente. Si una aplicación proporciona una estructura MQCNO, el cliente realiza un duplicado para transferirla a la salida donde se modifica. El cliente conserva la propiedad de MQCNO.

Un MQCD al que se hace referencia en MQCNO tiene prioridad sobre cualquier definición de conexión proporcionada mediante la matriz. El cliente utiliza la estructura MQCNO para conectarse con el gestor de colas y los demás se ignoran.

pCompCode
Tipo: PMQLONG entrada/salida

Código de terminación.

Puntero a un MQLONG que recibe el código de terminación de salidas. Tiene que ser uno de los valores siguientes:

- `MQCC_OK` - Terminación satisfactoria
- `MQCC_WARNING` -Aviso (terminación parcial)
- `MQCC_FAILED` -La llamada ha fallado

pReason
Tipo: PMQLONG entrada/salida

Razón que complementa a pCompCode.

Puntero a un MQLONG que recibe el código de razón de salida. Si el código de terminación es MQCC_OK, el único valor válido es:

- MQRC_NONE - (0, x'000') No hay ninguna razón sobre la que informar.

Si el código de terminación es MQCC_FAILED o MQCC_WARNING, la función de salida puede establecer el campo de código de razón a cualquier valor de MQRC_* válido.

Invocación C

```
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*/
```

Desarrollo y compilación de una salida de publicación

Se puede configurar una salida de publicación en el gestor de colas para cambiar el contenido de un mensaje publicado antes de que lo reciban los suscriptores. También se puede cambiar la cabecera del mensaje o no entregar el mensaje a una suscripción.

Nota: Las salidas de publicación no están soportadas en z/OS.

Se puede utilizar la salida de publicación para inspeccionar y modificar los mensajes entregados a los suscriptores:

- Examinar el contenido de un mensaje publicado en cada suscriptor.
- Modificar el contenido de un mensaje publicado en cada suscriptor.
- Modificar la cola en la que se coloca el mensaje.
- Detener la entrega de un mensaje a un suscriptor.

Desarrollo de una salida de publicación

Siga los pasos indicados en “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960 para obtener ayuda en el desarrollo y compilación de una salida.

El proveedor de la salida de publicación define lo que hace la salida. No obstante, la salida tiene que seguir las reglas definidas en MQPSXP.

IBM MQ no proporciona una implementación del punto de entrada MQ_PUBLISH_EXIT. Sí proporciona una declaración typedef en C. Utilice la declaración typedef para declarar los parámetros a una salida escrita por el usuario correctamente. En el ejemplo siguiente se ilustra cómo utilizar la declaración typedef:

```
#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 */
}
```

La salida de publicación se ejecuta en el proceso del gestor de colas, como resultado de las operaciones siguientes:

- Una operación de publicación en la que se entrega un mensaje a uno o más suscriptores.
- Una operación de suscripción en la que se entregan uno o varios mensajes retenidos.
- Una operación de solicitud de suscripción en la que se entregan uno o más mensajes retenidos.

Si se invoca la salida de publicación de una conexión, la primera vez que se invoca se establece un código *ExitReason* de MQXR_INIT. Antes de que la conexión se desconecte después de utilizar una salida de publicación, se llama a la salida con un código *ExitReason* de MQXR_TERM.

Si la salida de publicación está configurada, pero no se puede cargar cuando se inicia el gestor de colas, las operaciones de mensajes de publicación/suscripción se inhiben en dicho gestor de colas. Hay que solucionar el problema o reiniciar el gestor de colas para que la mensajería de publicación/suscripción se vuelva a habilitar.

Cada conexión de IBM MQ que requiera la salida de publicación podría no cargar o inicializar la salida. Si la salida no se puede cargar o inicializar, las operaciones de publicación/suscripción que requieran dicha salida quedarán inhabilitadas en esa conexión. Las operaciones fallan con el código de razón de IBM MQ MQRC_PUBLISH_EXIT_ERROR.

El contexto en el que invoca la salida de publicación es la conexión que efectúa una aplicación con el gestor de colas. El gestor de colas mantiene un área de datos de usuario por cada conexión que realiza operaciones de publicación. La salida puede conservar información en el área de datos de usuario en cada conexión.

Una salida de publicación puede utilizar algunas llamadas MQI. Solo puede utilizar las llamadas MQI que manipulan las propiedades del mensaje. Estas llamadas son:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Si la salida de publicación cambia el gestor de colas de destino o el nombre de cola, no se lleva a cabo ninguna comprobación de autorización nueva.

Compilación de una salida de publicación

La salida de publicación es una biblioteca cargada dinámicamente; es como una salida de canal. Para obtener información sobre la compilación de salidas, consulte “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960.

Ejemplo de salida de publicación

El programa de salida de ejemplo se llama amqspse0.c. Escribe un mensaje diferente en un archivo de registro en función de si la salida se invoca para una operación de inicialización, publicación o terminación. También ilustra el uso del campo de área de usuario de salida para asignar y liberar almacenamiento de forma adecuada.

Configuración de una salida de publicación

Hay que definir determinados atributos para configurar una salida de publicación.

En Windows y Linux, puede utilizar el explorador de IBM MQ para definir los atributos. Los atributos se definen en la página de propiedades del gestor de colas, bajo Publicación/suscripción.

Para configurar la salida de publicación en el archivo qm.ini en los sistemas AIX and Linux, cree una stanza llamada PublishSubscribe. La stanza PublishSubscribe tiene los atributos siguientes:

PublishExitPath=[path]|*nombre_módulo*
Nombre y ruta del módulo que contiene el código de salida de publicación. La longitud máxima de este campo es MQ_EXIT_NAME_LENGTH. El valor predeterminado es sin salida de publicación.

PublishExitFunction= *nombre_función*
Nombre del punto de entrada de la función al módulo que contiene el código de salida de publicación. La longitud máxima de este campo es MQ_EXIT_NAME_LENGTH.

IBM i En IBM i, si se utiliza el programa, omita PublishExitFunction.

PublishExitData= *cadena*
Si el gestor de colas invoca una salida de publicación, pasa una estructura MQPSXP como entrada. Los datos especificados en el atributo **PublishExitData** se proporcionan en el campo *ExitData* de la estructura. La cadena puede ser tener una longitud máxima de MQ_EXIT_DATA_LENGTH caracteres. El valor predeterminado es de 32 caracteres en blanco.

Escritura y compilación de salidas de carga de trabajo de clúster

Escriba un programa de salida de carga de trabajo de clúster para personalizar la gestión de la carga de trabajo de clústeres. Podría tener en cuenta el coste de utilizar un canal en diferentes momentos del día, o el contenido del mensaje, al direccionar mensajes. Estos son factores que no son considerados por el algoritmo de la gestión de carga de trabajo estándar.

En la mayoría de los casos, el algoritmo de gestión de carga de trabajo es suficiente para sus necesidades. No obstante, para que pueda proporcionar su propio programa de salida de usuario para personalizar la gestión de la carga de trabajo, IBM MQ incluye una salida de usuario, la salida de carga de trabajo del clúster.

Es posible que tenga alguna información concreta sobre la red o los mensajes que podría utilizar para influir en el equilibrio de carga de trabajo. Probablemente, sepa cuáles son los canales de alta capacidad o las rutas de red baratas, o podría desear direccionar los mensajes en función de su contenido. Puede optar por escribir un programa de salida de carga de trabajo de clúster, o utilizar uno proporcionado por un tercero.

Se invoca la salida de carga de trabajo del clúster cuando se accede a una cola de clúster. Se invoca mediante MQOPEN, MQPUT1 y MQPUT.

El gestor de colas de destino seleccionado en el momento MQOPEN se fija si se ha especificado MQOO_BIND_ON_OPEN. En este caso, la salida sólo se ejecuta una vez.

Si el gestor de colas de destino no se fija en el momento MQOPEN, el gestor de colas de destino se elige en el momento de la llamada de MQPUT. Si el gestor de colas de destino no está disponible, o falla mientras el mensaje sigue en la cola de transmisión, se vuelve a llamar a la salida. Se selecciona un nuevo gestor de colas de destino. Si el canal de mensajes falla durante la transferencia del mensaje y se restituye el mensaje, se selecciona un nuevo gestor de colas de destino.

Multi En Multiplatforms, el gestor de colas carga la nueva salida de la carga de trabajo del clúster la próxima vez que se inicia el gestor de colas.

Si la definición del gestor de colas no contiene un nombre de programa de salida de carga de trabajo de clúster, no se llama a la salida de carga de trabajo de clúster.

Se pasan distintos datos a una salida de carga de trabajo de clúster en la estructura de parámetro de salida, MQWXP:

- La estructura de definición de mensaje, MQMD.
- El parámetro de longitud de mensaje.
- Una copia del mensaje, o parte del mensaje.

En plataformas no de z/OS, si utiliza CLWLMode=FAST, cada proceso del sistema operativo carga su propia copia de la salida. Diferentes conexiones con el gestor de colas pueden provocar que se invoquen distintas copias de la salida. Si la salida se ejecuta en la modalidad segura predeterminada, CLWLMode=SAFE, una sola copia de la salida se ejecuta en su propio proceso separado.

Escribir salidas de carga de trabajo de clúster

z/OS Para obtener más información sobre cómo escribir salidas de carga de trabajo de clúster para z/OS, consulte la sección “Programación de salida de carga de trabajo de clúster para IBM MQ for z/OS” en la página 1024.

A partir de la IBM MQ 9.1.0, las salidas de la carga de trabajo del clúster se ejecutan en el espacio de direcciones del iniciador del canal, y no en el espacio de direcciones del gestor de colas. Si tiene una salida de carga de trabajo del clúster, debe eliminar la sentencia CSQXLIB DD en el procedimiento de tarea iniciada del gestor de colas y añadir el conjunto de datos que contiene la salida de carga de trabajo del clúster a la concatenación CSQXLIB en el procedimiento de tarea iniciada del iniciador del canal.

Multi En Multiplatforms, las salidas de carga de trabajo del clúster no deben utilizar llamadas MQI. En otros aspectos, las reglas para escribir y compilar las programas de salida de carga de trabajo de clúster son como las reglas que se aplican a los programas de salida de canal. Siga los pasos de la sección “Escritura de salidas y servicios instalables en AIX, Linux, and Windows” en la página 960, y utilice el programa de ejemplo de la sección “Salida de carga de trabajo de clúster de ejemplo” en la página 1023 como ayuda para escribir y compilar la salida.

Si desea más información sobre las salidas de canal, consulte “Cómo escribir programas de salida de canal” en la página 989.

Configuración de salidas de carga de trabajo de clúster

Puede asignar nombres a las salidas de carga de trabajo de clúster en la definición de gestor de colas, especificando el atributo de salida de carga de trabajo de clúster en el mandato `ALTER QMGR`. Por ejemplo:

```
ALTER QMGR CLWLEXIT(myexit)
```

Referencia relacionada

Llamada de salida de carga de trabajo del clúster y estructuras de datos

Salida de carga de trabajo de clúster de ejemplo

IBM MQ incluye un programa de salida de carga de trabajo de clúster de ejemplo. Puede copiar el ejemplo y utilizarlo como base para sus propios programas.

z/OS **IBM MQ for z/OS**

El programa de salida de carga de trabajo de clúster de ejemplo se proporciona en Assembler y en C. La versión de Assembler se denomina CSQ4BAF1 y se puede encontrar en la biblioteca `thlqual.SCSQASMS`. La versión en C se llama CSQ4BCF1 y se puede encontrar en la biblioteca `thlqual.SCSQC37S`. `thlqual` es el calificador de alto nivel de la biblioteca de destino para los conjuntos de datos de IBM MQ en la instalación.

Multi **IBM MQ for Multiplatforms**

El programa de salida de carga de trabajo de clúster de ejemplo se proporciona en C y se denomina `amqswlm0.c`. Se puede encontrar en:

Tabla 144. Ubicación del programa de salida de carga de trabajo de clúster de ejemplo para multiplataformas

Plataforma	Vía de acceso de archivo
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`

Tabla 144. Ubicación del programa de salida de carga de trabajo de clúster de ejemplo para multiplataformas (continuación)	
Plataforma	**Vía de acceso de archivo**
IBM i IBM i	La biblioteca qmqm

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Esta salida de ejemplo direcciona todos los mensajes a un determinado gestor de colas, a menos que ese gestor de colas no esté disponible. Reacciona a la anomalía del gestor de colas direccionando los mensajes a otro gestor de colas.

Indique a qué gestor de colas desea que se envíen los mensajes. Suministre el nombre del canal de clúster receptor en el atributo CLWLDATA en la definición del gestor de colas. Por ejemplo:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Para habilitar la salida, especifique la vía de acceso completa y el nombre en el atributo CLWLEXIT:

Linux AIX En AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows En Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS En z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

donde x es 'A' o 'C', dependiendo del lenguaje de programación de la versión que esté utilizando.

IBM i En IBM i, utilice cualquiera de los mandatos siguientes:

- Utilice el mandato MQSC:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  Tanto el nombre del programa como el nombre de la biblioteca ocupan 10 caracteres y se rellenan con blancos a la derecha si es necesario.
- Utilice el mandato CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

Ahora, en lugar de utilizar el algoritmo de gestión de carga de trabajo proporcionado, IBM MQ llama a esta salida para direccionar todos los mensajes al gestor de colas elegido.

z/OS *Programación de salida de carga de trabajo de clúster para IBM MQ for z/OS*

Las salidas de carga de trabajo de clúster se invocan como si lo hiciera un mandato de z/OS **LINK**. Las salidas están sujetas a una serie de reglas de programación estrictas. Evite utilizar la mayoría de los mandatos SVC que conllevan esperas, o utilizar una STAE o ESTAE en una salida de carga de trabajo.

Las salidas de carga de trabajo de clúster se invocan como si lo hiciera un mandato de z/OS **LINK** en:

- Estado de programa anómalo no autorizado
- Modalidad de control de espacio de direcciones primaria
- Modalidad de memoria no cruzada
- Modalidad de registro sin acceso
- Modalidad de direccionamiento de 31 bits
- Clave de almacenamiento 8
- Máscara de clave de programa 8
- Clave TCB 8

Coloque los módulos editados por enlace en el conjunto de datos especificado por la sentencia CSQXLIB DD del procedimiento de tarea iniciada del iniciador de canal. Los nombres de los módulos de carga se especifican como los nombres de salida de carga de trabajo en la definición del gestor de colas.

Al escribir salidas de carga de trabajo para IBM MQ for z/OS, se aplican las reglas siguientes:

- Debe escribir salidas en assembler o C. Si utiliza C, debe ajustarse al entorno de programación de sistemas C para salidas del sistema, que se describe en la publicación *z/OS C/C++ Programming Guide*, SC09-4765.
- Si utiliza la llamada MQXCLWLN, enlace con CSQMFCLW, proporcionado en *thlqual*.SCSQLOAD.
- Las salidas se cargan desde las bibliotecas no autorizadas definidas mediante una sentencia CSQXLIB DD. Siempre y cuando CSQXLIB tenga DISP=SHR, las salidas se pueden actualizar mientras el gestor de colas está en ejecución, con la nueva versión utilizada en la siguiente hebra MQCONN que inicia el gestor de colas.
- Las salidas deben volver a entrar y deben poder ejecutarse en cualquier lugar del almacenamiento virtual.
- Al volver, las salidas deben restablecer el entorno al que había en la entrada.
- Las salidas deben liberar cualquier almacenamiento obtenido, o asegúrese de liberar el almacenamiento mediante una invocación de salida posterior.
- No se permiten llamadas MQI.
- Las salidas no deben utilizar ningún servicio del sistema que pueda ocasiona una espera, ya que una espera degrada seriamente el rendimiento del gestor de colas. Por lo tanto, en general, evite un SVC, PC o I/O.
- Las salidas no deben emitir una ESTAE o SPIE, excepto en cualquier subtarea que adjunten.

Nota: No hay ninguna restricción absoluta sobre lo que puede hacer en una salida. Sin embargo, la mayoría de los SVC conllevan esperas, por lo que debe evitarlos, salvo en el caso de los siguientes mandatos:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

No utilice ESTAE y ESPIE porque su manejo de errores podría interferir con el manejo de errores realizado por IBM MQ. Es posible que IBM MQ no pueda recuperarse de un error, o que el programa de salida no reciba toda la información de error.

El parámetro del sistema EXITLIM limita la cantidad de tiempo durante el que puede ejecutarse una salida. El valor predeterminado para EXITLIM es 30 segundos. Si ve el código de retorno MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA', la salida podría estar en un bucle. Si piensa que la salida necesita más de 30 segundos para completarse, aumente el valor de EXITLIM.

Creación de una aplicación procedimental

Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

AIX Creación de una aplicación procedimental en AIX

Las publicaciones de AIX describen cómo crear aplicaciones ejecutables desde los programas que escribe.

En este tema, se describen las tareas adicionales y los cambios en las tareas estándar que debe realizar cuando crea aplicaciones de IBM MQ for AIX para que se ejecuten en AIX. Se da soporte a C, C++ y COBOL. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++.

Las tareas que debe realizar para crear una aplicación ejecutable utilizando IBM MQ for AIX varían con el lenguaje de programación en el que se escribe el código fuente. Además de codificar las llamadas MQI en el código fuente, debe añadir las sentencias de idioma adecuadas para incluir los archivos de inclusión de IBM MQ for AIX para el lenguaje que está utilizando. Familiarícese con el contenido de estos archivos. Consulte “Archivos de definición de datos de IBM MQ” en la página 736 para obtener una descripción completa.

Cuando ejecute aplicaciones de cliente o servidor con hebras, establezca la variable de entorno AIXTHREAD_SCOPE=S.

AIX *Preparación de programas C en AIX*

Este tema contiene información sobre el enlace de las bibliotecas necesarias para preparar programas C en AIX.

Se proporcionan programas C precompilados en el directorio *MQ_INSTALLATION_PATH* `/samp/bin`. Utilice el compilador ANSI y ejecute los mandatos siguientes. Para obtener más información sobre la programación de aplicaciones de 64 bits, consulte Estándares de codificación en plataformas de 64 bits.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para aplicaciones de 32 bits:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

donde `amqsput0` es un programa de ejemplo.

Para aplicaciones de 64 bits:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

donde `amqsput0` es un programa de ejemplo.

V 9.3.5 Para aplicaciones de 32 bits que utilizan el compilador XLC 17:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

donde `amqsput0` es un programa de ejemplo.

V 9.3.5 Para aplicaciones de 64 bits que utilizan el compilador XLC 17:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

donde `amqsput0` es un programa de ejemplo.

Si utiliza el compilador C/C++ VisualAge para programas C++, debe incluir la opción `-q namemangling=v5` para obtener todos los símbolos de IBM MQ resueltos al enlazar las bibliotecas.

Si desea utilizar los programas en una máquina que solo tiene IBM MQ MQI client for AIX instalado, vuelva a compilar los programas para enlazarlos con la biblioteca de cliente (`-lmqic`) en su lugar.

Enlazar bibliotecas

Necesita las bibliotecas siguientes:

- Enlace sus programas con la biblioteca adecuada proporcionada por IBM MQ.

  En un entorno sin hebras, enlace con una de las bibliotecas siguientes:

Archivo de biblioteca	Tipo programa/salida
libmqm.a	Servidor en C
libmqic.a & libmqm.a	Cliente en C

  En un entorno con hebras, enlace con una de las bibliotecas siguientes:

Archivo de biblioteca	Tipo programa/salida
libmqm_r.a	Servidor en C
libmqic_r.a & libmqm_r.a	Cliente en C

  Por ejemplo, para crear una aplicación IBM MQ simple con hebras desde una unidad de compilación individual, ejecute los mandatos siguientes.

  Para aplicaciones de 32 bits:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  donde `amqsput0` es un programa de ejemplo.

  Para aplicaciones de 64 bits:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  donde `amqsput0` es un programa de ejemplo.

  V 9.3.5 Para aplicaciones de 32 bits que utilizan el compilador XLC 17:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

  donde `amqsput0` es un programa de ejemplo.

  V 9.3.5 Para aplicaciones de 64 bits que utilizan el compilador XLC 17:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  donde `amqsput0` es un programa de ejemplo.

  Si desea utilizar los programas en una máquina que solo tiene IBM MQ MQI client for AIX instalado, vuelva a compilar los programas para enlazarlos con la biblioteca de cliente (`-lmqic`) en su lugar.

  Nota:

  1. No se puede enlazar a más de una biblioteca. Es decir, no se puede enlazar a una biblioteca con hebras ni a una biblioteca sin hebras al mismo tiempo.
  2. Si está escribiendo un servicio instalable (consulte Administración IBM MQ para obtener más información), necesitará enlazar con la biblioteca `libmqmzf.a` en una aplicación sin hebras y con la biblioteca `libmqmzf_r.a` en una aplicación con hebras.
  3. Si va a producir una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como IBM TXSeries, Encina o BEA Tuxedo, necesitará enlazar con las bibliotecas `libmqmxa.a` (o `libmqmxa64.a` si su gestor de transacciones trata el tipo 'long' como de 64 bits) y

libmqz.a en una aplicación sin hebras y con las bibliotecas libmqmxa_r.a (o libmqmxa64_r.a) y libmqz_r.a en una aplicación con hebras.

4. Necesita enlazar las aplicaciones de confianza con las bibliotecas de IBM MQ con hebras. Sin embargo, sólo se puede conectar una hebra en una aplicación de confianza en sistemas IBM MQ for AIX or Linux a la vez.
5. Debe enlazar las bibliotecas de IBM MQ antes que ninguna otra biblioteca del producto.

AIX *Preparación de programas COBOL en AIX*

Use esta información cuando prepare programas COBOL en AIX usando IBM COBOL Set y Micro Focus COBOL.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

- Los libros de copia COBOL de 32 bits se instalan en el directorio siguiente:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  y los enlaces simbólicos se crean en:

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- Los libros de copias COBOL de 64 bits se instalan en el directorio siguiente:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

En los ejemplos siguientes, establezca la variable de entorno **COBCPY** a:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

para aplicaciones de 32 bits, y:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

en aplicaciones de 64 bits.

Hay que enlazar el programa con uno de los siguientes archivos de biblioteca:

Archivo de biblioteca	Tipo programa/salida
libmqmcb.a	Servidor de COBOL (aplicación sin hebras)
libmqmcb_r.a	Servidor para COBOL (aplicación con hebras)
libmqicb.a	Cliente de COBOL (aplicación sin hebras)
libmqicb_r.a	Cliente para COBOL (aplicación con hebras)

Se puede usar el compilador IBM COBOL Set o el compilador Micro Focus COBOL, dependiendo del programa:

- Los programas que empiezan por amqm son adecuados para el compilador Micro Focus COBOL y
- los programas que empiezan por amq0 son adecuados para cualquiera de los dos compiladores.

Preparación de programas COBOL con IBM COBOL Set para AIX

Se proporcionan ejemplos de programas en COBOL en IBM MQ. Para compilar un programa de este tipo, especifique el correspondiente comando de la lista siguiente:

Aplicación de servidor sin hebras de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

Aplicación cliente sin hebras de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

Aplicación de servidor con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Aplicación cliente con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Aplicación de servidor sin hebras de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

Aplicación cliente sin hebras de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

Aplicación de servidor con hebras de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Aplicación cliente con hebras de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Preparación de programas COBOL utilizando Micro Focus COBOL

Establezca las variables de entorno antes de compilar el programa tal como se indica a continuación:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Para compilar un programa COBOL de 32 bits con Micro Focus COBOL, ejecute:

- Servidor para COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Cliente para COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Servidor de COBOL con hilos

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Cliente COBOL con hilos

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Para compilar un programa COBOL de 64 bits utilizando Micro Focus COBOL, especifique:

- Servidor para COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Cliente para COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Servidor de COBOL con hilos

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Cliente COBOL con hilos

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

donde amqminqx es un programa de ejemplo

Consulte la documentación de Micro Focus COBOL para obtener una descripción de las variables de entorno que hay que configurar.

AIX *Preparación de programas de aplicación de CICS en AIX*

Utilice esta información al preparar programas de CICS en AIX.

Utilice módulos *XA switch* para enlazar CICS con IBM MQ. Para obtener más información sobre la estructura de conmutación XA, consulte Las estructuras de conmutación XA.

El archivo de código fuente de ejemplo se proporciona para permitirle desarrollar los conmutadores XA para otros mensajes de transacción. El nombre del módulo de carga del conmutador proporcionado se lista en Tabla 145 en la página 1030.

Tabla 145. Código esencial para programas de aplicación de CICS en AIX: rutina de inicialización XA

Descripción	**C (fuente)**	**C (exec) - añádase a XAD.Stanza**
Rutina de inicialización XA	amqzscix.c	*amqzsc* - CICS para AIX

Utilice la versión preconstruida del archivo de carga conmutada de IBM MQ *amqzsc*, que se proporciona con el producto.

Enlace siempre las transacciones C con la biblioteca IBM MQ de hebras seguras *libmqm_r.a.*, y las transacciones COBOL con la biblioteca COBOL *libmqmcb_r.a.*.

Puede encontrar más información sobre el soporte de transacciones de CICS en la publicación Administración IBM MQ IBM MQ System Administration Guide.

AIX *Soporte de TXSeries CICS*

IBM MQ en AIX da soporte a TXSeries CICS utilizando la interfaz XA. Asegúrese de que las aplicaciones CICS estén enlazadas con la versión de hebras de la biblioteca de IBM MQ.

Puede ejecutar programas CICS utilizando el conjunto IBM COBOL para AIX o Micro Focus COBOL. Las secciones siguientes describen la diferencia entre ejecutar programas CICS en el conjunto IBM COBOL para AIX y Micro Focus COBOL.

Escriba programas de IBM MQ que se carguen en la misma región de CICS en C o en COBOL. No puede hacer una combinación de llamadas MQI de C y COBOL en la misma región de CICS. La mayoría de las llamadas MQI del segundo lenguaje utilizado fallan, y se emite un código de razón de `MQRC_HOBJ_ERROR`.

Preparación de programas CICS COBOL utilizando el conjunto IBM COBOL para AIX

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para utilizar IBM COBOL, siga estos pasos:

1. Exporte la variable de entorno siguiente:

   ```
   export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                   -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                   -e _iwz_cobol_main  \
   ```

   donde `LIB` es una directiva del compilador.
2. Convierta, compile y enlace el programa escribiendo el mandato siguiente:

   ```
   cicstcl -l IBMCOB yourprog.ccp
   ```

Preparación de programas CICS COBOL utilizando Micro Focus COBOL

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para utilizar Micro Focus COBOL, siga estos pasos:

1. Añada el módulo de bibliotecas de tiempo de ejecución de IBM MQ COBOL a la biblioteca de tiempo de ejecución utilizando el mandato siguiente:

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Nota: Con `cicsmkcobol`, IBM MQ no permite realizar llamadas MQI en el lenguaje de programación C desde su aplicación COBOL.

   Si las aplicaciones de que dispone tienen este tipo de llamadas, se le recomienda que mueva dichas funciones desde las aplicaciones COBOL a su propia biblioteca, por ejemplo, `myMQ.so`. Después de trasladar las funciones, no incluya la biblioteca de IBM MQ, `libmqmcbrt.o`, cuando cree la aplicación COBOL para CICS.

   Adicionalmente, si su aplicación COBOL no realiza ninguna llamada COBOL MQI, no enlace `libmqmz_r` con `cicsmkcobol`.

   Esto crea el archivo de método de lenguaje COBOL de Micro Focus y habilita la biblioteca COBOL de tiempo de ejecución de CICS para llamar a sistemas IBM MQ for AIX or Linux .

   Nota: Ejecute `cicsmkcobol` solo cuando instale uno de los productos siguientes:

   - Versión o release nuevo de Micro Focus COBOL
   - Versión o release nuevo de CICS para AIX
   - Versión o release nuevo de cualquier producto de base de datos soportado (solo para las transacciones COBOL)
   - Versión o release nuevo de IBM MQ
2. Exporte la variable de entorno siguiente:

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. Convierta, compile y enlace el programa escribiendo el mandato siguiente:

```
cicstcl -l COBOL -e yourprog.ccp
```

Preparación de programas C de CICS

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Cree programas C de CICS utilizando los recursos CICS estándar:

1. Exporte ***una*** de las variables de entorno siguientes:
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export USERLIB
2. Convierta, compile y enlace el programa escribiendo el mandato siguiente:

```
cicstcl -l C amqscic0.ccs
```

Transacción C de CICS de ejemplo

AMQSCIC0.CCS proporciona código fuente C de ejemplo para una transacción AIX IBM MQ. La transacción lee mensajes de la cola de transmisión SYSTEM.SAMPLE.CICS.WORKQUEUE en el gestor de colas predeterminado y los coloca en la cola local con un nombre de cola que está contenido en la cabecera de transmisión del mensaje. Las anomalías se envían a la cola SYSTEM.SAMPLE.CICS.DLQ. Utilice el script MQSC AMQSCIC0.TST de ejemplo para crear estas colas y las colas de entrada de ejemplo.

IBM i Creación de una aplicación procedimental en IBM i

Las publicaciones de IBM i describen cómo crear aplicaciones ejecutables desde los programas que escribe, para ejecutarse con IBM i en los sistemas iSeries o System i.

En este tema, se describen las tareas adicionales y los cambios en las tareas estándar que debe realizar cuando crea aplicaciones procedimentales de IBM MQ for IBM i para que se ejecuten en sistemas IBM i. Los lenguajes de programación COBOL, C, C++, Java y RPG están soportados. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++. Para obtener información sobre cómo preparar los programas Java, consulte Utilización de IBM MQ classes for Java.

Las tareas que debe realizar para crear una aplicación ejecutable de IBM MQ for IBM i dependen del lenguaje de programación en el que se escriba el código fuente. Además de codificar las llamadas MQI en el código fuente, debe añadir las sentencias de idioma adecuadas para incluir los archivos de definición de datos de IBM MQ for IBM i para el lenguaje que está utilizando. Familiarícese con el contenido de estos archivos. Consulte “Archivos de definición de datos de IBM MQ” en la página 736 para obtener una descripción completa.

IBM i *Preparación de programas C en IBM i*

IBM MQ for IBM i da soporte a mensajes de hasta 100 MB de tamaño. Los programas de aplicación escritos en ILE C, que dan soporte a los mensajes de IBM MQ de más de 16 MB, tienen que utilizar la opción de compilador de `Teraespacio` para asignar suficiente memoria para estos mensajes.

Para obtener más información sobre las opciones del compilador C, consulte *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Para compilar un módulo C, puede utilizar el mandato **CRTCMOD**de IBM i . Asegúrese de que la biblioteca que contiene los archivos de inclusión (QMQM) esté en la lista de bibliotecas cuando realice la compilación.

A continuación, debe enlazar la salida del compilador con el programa de servicio utilizando el mandato **CRTPGM** .

Tabla 146. Ejemplo de CRTPGM en entornos sin hebras y con hebras

Tipo de entorno	Mandato	Tipo programa/salida
Entorno sin hebras	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM)`	Servidor o cliente para C
Entorno con hebras	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM_R)`	Servidor o cliente para C

donde *pgmname* es el nombre del programa.

La Tabla 147 en la página 1033 lista las bibliotecas que son necesarias al preparar programas C en IBM i en un entorno sin hebras y un entorno con hebras.

Tabla 147. Bibliotecas necesarias para entornos sin hebras y con hebras

Tipo de entorno	Archivo de biblioteca	Tipo programa/salida
Entorno sin hebras	LIBMQM	Servidor en C
	LIBMQIC y LIBMQM	Cliente en C
Entorno con hebras	LIBMQM_R	Servidor en C
	LIBMQIC_R y LIBMQM_R	Cliente en C

IBM i *Preparación de programas COBOL en IBM i*

Obtenga información sobre la preparación de programas de COBOL en IBM i y el método para acceder a la MQI desde el programa de COBOL.

Acerca de esta tarea

Para acceder a la MQI desde los programas de COBOL, IBM MQ for IBM i proporciona una interfaz de llamada a procedimiento enlazado proporcionada por los programas de servicio. Esto proporciona acceso a todas las funciones de MQI en IBM MQ for IBM i y soporte para las aplicaciones con hebras. Esta interfaz solo se puede utilizar con el compilador COBOL ILE.

La sintaxis de CALL de COBOL estándar se utiliza para acceder a las funciones de MQI.

Los archivos de copia COBOL que contienen las constantes nombradas y las definiciones de estructura para su uso con la MQI se encuentran en el archivo fuente QMQM/QCBLLESRC.

Los archivos de copia COBOL utilizan el carácter de comilla simple (') como delimitador de cadenas. Los compiladores COBOL de IBM i presuponen que el delimitador es la comilla ("). Para evitar que los compiladores generen mensajes de aviso, especifique `OPTION (*APOST)` en los mandatos **CRTCBLPGM**, **CRTBNDCBL**o **CRTCBLMOD**.

Para hacer que el compilador acepte las comillas simples (') como delimitador de cadenas en los archivos de copia COBOL, use la opción de compilador \APOST.

Nota: La interfaz de llamada dinámica no se proporciona en IBM MQ 9.0 o posterior.

Para utilizar la interfaz de llamada a procedimiento enlazado, complete los pasos siguientes.

Procedimiento

1. Cree un módulo utilizando el compilador **CRTCBLMOD** y especificando el parámetro:

```
LINKLIT(*PRC)
```

2. Utilice el mandato **CRTPGM** para crear el objeto de programa, especificando el parámetro adecuado:

   En aplicaciones sin hebras:

```
BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
```

   Para las aplicaciones con hebras:

```
BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
```

   Nota: Excepto para los programas creados utilizando el compilador COBOL de V4R4 ILE y que contengan la opción THREAD(SERIALIZE) en la sentencia PROCESS, los programas COBOL no deben utilizar las bibliotecas IBM MQ con hebras. Incluso si un programa COBOL se hace de hebra segura (thread-safe) de esta manera, tenga cuidado al diseñar la aplicación, porque THREAD(SERIALIZE) fuerza la serialización de los procedimientos COBOL a nivel de módulo y puede penalizar el rendimiento general.

   Consulte las publicaciones *WebSphere Development Studio: ILE COBOL Programmer's Guide* y *WebSphere Development Studio: ILE COBOL Reference* para obtener más información.

   Para obtener más información sobre la compilación de una aplicación de CICS, consulte la publicación *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *Preparación de programas CICS en IBM i*

Obtenga información acerca de los pasos necesarios para la preparación de los programas CICS en IBM i.

Para crear un programa que incluya sentencias EXEC de CICS y llamadas MQI, realice estos pasos:

1. Si es necesario, prepare las correlaciones utilizando el mandato CRTCICSMAP.
2. Convierta los mandatos EXEC CICS en sentencias de lenguaje nativo. Utilice el mandato CRTCICSC para un programa C. Utilice el mandato CRTCICSCBL para un programa COBOL.

   Incluya `CICSOPT(*NOGEN)` en el mandato CRTCICSC o CRTCICSCBL. Esto detiene el proceso para que pueda incluir los programas de servicio CICS e IBM MQ adecuados. Este mandato coloca el código, de forma predeterminada, en QTEMP/QACYCICS.
3. Compile el código fuente utilizando el mandato CRTCMOD, para un programa C, o el mandato CRTCBLMOD, para un programa COBOL.
4. Utilice CRTPGM para enlazar el código compilado con los programas de servicio CICS e IBM MQ adecuados. Esto crea el programa ejecutable.

El siguiente es un ejemplo de este tipo de código. Compila el programa de ejemplo de CICS:

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *Preparación de los programas RPG en IBM i*

Si está utilizando IBM MQ for IBM i, puede escribir sus aplicaciones en RPG.

Para obtener más información, consulte “Desarrollo de programas IBM MQ en RPG (solo IBM i)” en la página 1084 e IBM i Application Programming Reference (ILE/RPG).

IBM i *Consideraciones sobre la programación SQL para IBM i*

Obtenga información sobre los pasos necesarios al crear una aplicación en IBM i utilizando SQL.

Si el programa contiene sentencias EXEC SQL y llamadas MQI, siga estos pasos:

1. Convierta los mandatos EXEC SQL en sentencias de lenguaje nativo. Utilice el mandato CRTSQLCI para un programa C. Utilice el mandato CRTSQLCBLI para un programa COBOL.

   Incluya `OPTION(*NOGEN)` en el mandato CRTSQLCI o CRTSQLCBLI. Esto detiene el proceso para permitirle incluir los programas de servicio de IBM MQ adecuados. Este mandato pone el código, de forma predeterminada, en QTEMP/QSQLTEMP.
2. Compile el código fuente utilizando el mandato CRTCMOD, para un programa C, o el mandato CRTCBLMOD, para un programa COBOL.
3. Utilice CRTPGM para enlazar el código compilado con los programas de servicio de IBM MQ adecuados. Esto crea el programa ejecutable.

A continuación se muestra un ejemplo de este código (compila un programa, SQLTEST, en la biblioteca, SQLUSER):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Creación de una aplicación procedimental en Linux

En esta información, se describen las tareas adicionales y los cambios en las tareas estándar que debe realizar cuando crea aplicaciones de IBM MQ para Linux para ejecutarlas.

Se da soporte a C y C++. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++.

Linux *Preparación de programas C en Linux*

Los programas C precompilados se proporcionan en el directorio *MQ_INSTALLATION_PATH*`/samp/bin` . Para crear un ejemplo a partir del código fuente, utilice el compilador `gcc`.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Trabaje en el entorno habitual. Para obtener más información acerca de cómo programar aplicaciones de 64 bits, consulte la sección Estándares de codificación en las plataformas de 64 bits.

Enlazar bibliotecas

En las tablas siguientes se listan las bibliotecas que son necesarias al preparar programas C en Linux.

- Necesitará enlazar sus programas con la biblioteca pertinente, proporcionada por IBM MQ.

  En un entorno sin hebras, enlace sólo a una de las bibliotecas siguientes:

Archivo de biblioteca	Tipo programa/salida
libmqm.so	Servidor en C
libmqic.so & libmqm.so	Cliente en C

  En un entorno con hebras, enlace sólo a una de las bibliotecas siguientes:

Archivo de biblioteca	Tipo programa/salida
libmqm_r.so	Servidor en C
libmqic_r.so & libmqm_r.so	Cliente en C

Nota:

1. No se puede enlazar a más de una biblioteca. Es decir, no se puede enlazar a una biblioteca con hebras ni a una biblioteca sin hebras al mismo tiempo.
2. Si escribe un servicio instalable (para obtener más información, consulte Administración IBM MQ), debe enlazar a la biblioteca `libmqmzf.so`.
3. Si produce una aplicación para coordinación externa mediante un gestor de transacciones compatible con XA, como IBM TXSeries Encina, o BEA Tuxedo, debe enlazar a `libmqmxa.so` (o `libmqmxa64.so` si su gestor de transacciones trata el tipo 'long' como de 64 bits) y a bibliotecas `libmqz.so` en una aplicación sin hebras y a bibliotecas `libmqmxa_r.so` (o `libmqmxa64_r.so`) y `libmqz_r.so` en una aplicación con hebras.
4. Debe enlazar las bibliotecas de IBM MQ antes que ninguna otra biblioteca del producto.

Linux *Creación de aplicaciones de 31 bits*

Este tema contiene ejemplos de los mandatos que se utilizan para crear programas de 31 bits en diversos entornos.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Aplicación de cliente en C de 31 bits sin hebras

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicación de cliente en C de 31 bits con hebras

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicación de servidor en C de 31 bits sin hebras

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicación de servidor en C de 31 bits con hebras

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicación de cliente en C++ de 31 bits sin hebras

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicación de cliente en C++ de 31 bits con hebras

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Aplicación de servidor en C++ de 31 bits sin hebras

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicación de servidor en C++ de 31 bits con hebras

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Salida de cliente en C de 31 bits sin hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Salida de cliente en C de 31 bits con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Salida de servidor en C de 31 bits sin hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Salida de servidor en C de 31 bits con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Creación de aplicaciones de 32 bits*

Este tema contiene ejemplos de los mandatos que se utilizan para crear programas de 32 bits en diversos entornos.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Aplicación de cliente en C de 32 bits sin hebras

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicación de cliente en C de 32 bits con hebras

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicación de servidor en C de 32 bits sin hebras

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicación de servidor en C de 32 bits con hebras

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicación de cliente en C++ de 32 bits sin hebras

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicación de cliente en C++ de 32 bits con hebras

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Aplicación de servidor en C++ de 32 bits sin hebras

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicación de servidor en C++ de 32 bits con hebras

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Salida de cliente en C de 32 bits sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Salida de cliente en C de 32 bits con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Salida de servidor en C de 32 bits sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Salida de servidor en C de 32 bits con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Compilación de aplicaciones de 64 bits*

Este tema contiene ejemplos de los comandos que se utilizan para crear programas de 64 bits en diversos entornos.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Aplicación cliente en C, 64 bits, sin hebras

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Aplicación cliente en C, 64 bits, con hebras

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Aplicación de servidor en C, 64 bits, sin hebras

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Aplicación de servidor en C, 64 bits, con hebras

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Aplicación cliente en C++, 64 bits, sin hebras

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación cliente en C++, 64 bits, con hebras

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

aplicación de servidor en C++, 64 bits, sin hebras

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicación de servidor en C++, 64 bits, con hebras

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Salida cliente en C , 64 bits, sin hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Salida cliente en C , 64 bits, con hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Salida de servidor en C, 64 bits, sin hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Salida de servidor en C, 64 bits, con hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *Preparación de programas COBOL en Linux*

Aprenda a preparar programas COBOL en Linux y a preparar programas COBOL utilizando IBM COBOL for Linux en x86 y Micro Focus COBOL.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

1. Los libros de copia COBOL de 32 bits se instalan en el directorio siguiente:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   y los enlaces simbólicos se crean en:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. En plataformas de 64 bits, los libros de copia COBOL de 64 bits se instalan en el directorio siguiente:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. En los ejemplos siguientes, establezca COBCPY en:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   para aplicaciones de 32 bits, y:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   para aplicaciones de 64 bits.

Tiene que enlazar el programa con uno de los siguientes:

Archivo de biblioteca	Tipo programa/salida
libmqmcb.so	Servidor para COBOL
libmqicb.so	Cliente para COBOL
libmqmcb_r.so	Servidor para COBOL (aplicación con hebras)

Archivo de biblioteca	Tipo programa/salida
libmqicb_r.so	Cliente para COBOL (aplicación con hebras)

Preparación de programas COBOL utilizando IBM COBOL for Linux en x86

Los programas COBOL de ejemplo se proporcionan con IBM MQ. Para compilar un programa de este tipo, especifique el correspondiente comando de la lista siguiente:

Aplicación de servidor sin hebras de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

Aplicación cliente sin hebras de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

Aplicación de servidor con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

Aplicación cliente con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

Preparación de programas COBOL utilizando Micro Focus COBOL

Establezca las variables de entorno antes de compilar el programa tal como se indica a continuación:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Para compilar un programa COBOL de 32 bits, donde esté soportado, utilizando Micro Focus COBOL, entre:

```
$ 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
```

Para compilar un programa COBOL de 64 bits utilizando Micro Focus COBOL, entre:

```
$ 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
```

donde `amqsput` es un programa de ejemplo

Consulte la documentación de Micro Focus COBOL para obtener una descripción de las variables de entorno que necesita.

Windows Creación de una aplicación procedimental en Windows

Las publicaciones de los sistemas Windows describen cómo crear aplicaciones ejecutables a partir de los programas que escriba.

Este tema describe las tareas adicionales y los cambios en las tareas estándar que debe realizar cuando crea aplicaciones IBM MQ for Windows que se ejecutan en Windows. Se da soporte a los lenguajes de programación C, C++, COBOL y Visual Basic. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++.

Las tareas que debe realizar para crear una aplicación ejecutable utilizando IBM MQ for Windows varían con el lenguaje de programación en el que se escribe el código fuente. Además de codificar las llamadas MQI en el código fuente, debe añadir las sentencias de idioma adecuadas para incluir los archivos de inclusión de IBM MQ for Windows para el lenguaje que está utilizando. Familiarícese con el contenido de estos archivos. Consulte "Archivos de definición de datos de IBM MQ" en la página 736 para obtener una descripción completa.

Windows *Creación de aplicaciones de 64 bits en Windows*

En IBM MQ for Windows se admiten aplicaciones de 32 bits y de 64 bits. Los archivos ejecutables y de biblioteca de IBM MQ se suministran en formularios de 32 bits y de 64 bits, utilice la versión adecuada en función de la aplicación con la que está trabajando.

Bibliotecas y archivos ejecutables

Las dos versiones de 32 bits y de 64 bits de las bibliotecas de IBM MQ se proporcionan en las ubicaciones siguientes:

Tabla 148. Ubicación de las bibliotecas de IBM MQ

Versión de biblioteca	**Directorio que contiene los archivos de biblioteca**
32 bits	*MQ_INSTALLATION_PATH* \Tools\Lib
64 bits	*MQ_INSTALLATION_PATH* \Tools\Lib64

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Tras la migración, las aplicaciones de 32 bits siguen funcionando con normalidad. Los archivos de 32 bits están en el mismo directorio que en las versiones anteriores del producto.

Si quiere crear una versión de 64 bits, debe asegurarse de que su entorno esté configurado para usar archivos de biblioteca en *MQ_INSTALLATION_PATH* \Tools\Lib64. Asegúrese de que la variable de entorno LIB no esté establecida para buscar en la carpeta que contiene las bibliotecas de 32 bits.

Windows *Preparación de programas C en Windows*

Trabaje en su entorno habitual de Windows; IBM MQ for Windows no requiere nada en especial.

Para obtener más información sobre la programación de aplicaciones de 64 bits, consulte Estándares de codificación en plataformas de 64 bits.

- Enlace sus programas con las bibliotecas adecuadas proporcionadas por IBM MQ:

Archivo de biblioteca	**Tipo programa/salida**
MQ_INSTALLATION_PATH \Tools\Lib\mqm.lib	Servidor para C de 32 bits
MQ_INSTALLATION_PATH \Tools\Lib\mqic.lib	Cliente para C de 32 bits
MQ_INSTALLATION_PATH \Tools\Lib\mqicxa.lib	Cliente para C de 32 bits con coordinación de transacciones
MQ_INSTALLATION_PATH \Tools\Lib64\mqm.lib	Servidor para C de 64 bits

Archivo de biblioteca	Tipo programa/salida
MQ_INSTALLATION_PATH \Tools\Lib64\mqic.lib	Cliente para C de 64 bits
MQ_INSTALLATION_PATH \Tools\Lib64\mqicxa.lib	Cliente para C de 64 bits con coordinación de transacciones

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

El mandato siguiente ofrece un ejemplo de compilación del programa de ejemplo amqsget0 (utilizando el compilador de Microsoft Visual C++).

Para aplicaciones de 32 bits:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Para aplicaciones de 64 bits:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

Nota:

- Si está escribiendo un servicio instalable (consulte Administración IBM MQ para obtener más información), necesitará enlazar con la biblioteca mqmzf.lib.
- Si va a producir una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA, como IBM TXSeries Encina o BEA Tuxedo, necesitará enlazar con la biblioteca mqmxa.lib o mqmxa.lib.
- Si está escribiendo una salida CICS, enlace con la biblioteca mqmcics4.lib.
- Debe enlazar las bibliotecas de IBM MQ antes que ninguna otra biblioteca del producto.

- Las DLL deben estar en la vía de acceso (PATH) que haya especificado.
- Si utiliza caracteres en minúsculas siempre que sea posible, puede pasar de IBM MQ for Windows a los sistemas IBM MQ for AIX or Linux , donde es necesario utilizar minúsculas.

Preparación de programas CICS y Transaction Server

AMQSCIC0.CCS proporciona código fuente C de ejemplo para una transacción CICS IBM MQ. Puede compilarlo utilizando los recursos estándares de CICS. Por ejemplo, para TXSeries para Windows 2000:

1. Establezca la variable de entorno (especifique el código siguiente en una línea):

   ```
   set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
   %CICS_IBMC_FLAGS%
   ```

2. Establezca la variable de entorno USERLIB:

   ```
   set USERLIB=MQM.LIB;%USERLIB%
   ```

3. Convierta, compile y enlace el programa de ejemplo:

   ```
   cicstcl -l IBMC amqscic0.ccs
   ```

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Esto se describe en la *Guía de programación de aplicaciones de Transaction Server para Windows NT (CICS) V4*.

Puede encontrar más información sobre las transacciones CICS de soporte en Administración IBM MQ.

Windows *Preparación de programas COBOL en Windows*

Utilice esta información para aprender a preparar programas COBOL en Windows y preparar programas de CICS y Transaction Server.

1. Los libros de copia COBOL de 32 bits se instalan en el directorio siguiente: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook`.
2. Los libros de copia COBOL de 64 bits se instalan en el directorio siguiente: *MQ_INSTALLATION_PATH* `\Tools\cobol\CopyBook64`
3. En los ejemplos siguientes, establezca CopyBook en:

   ```
   CopyBook
   ```

   para aplicaciones de 32 bits, y:

   ```
   CopyBook64
   ```

   para aplicaciones de 64 bits.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para preparar programas COBOL en sistemas Windows, enlace el programa a una de las bibliotecas siguientes proporcionadas por IBM MQ:

Archivo de biblioteca	**Tipo de salida o programa**
MQ_INSTALLATION_PATH `\Tools\Lib\mqmcb`	Servidor de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib\mqiccb`	Cliente de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqmcb`	Servidor de 64 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH `\Tools\Lib64\mqiccb`	Cliente de 64 bits para Micro Focus COBOL

Al ejecutar un programa en el entorno de cliente de MQI, asegúrese de que la biblioteca DOSCALLS aparece antes de cualquier biblioteca de COBOL o IBM MQ.

Preparación de programas COBOL utilizando Micro Focus COBOL

Vuelva a enlazar los programas IBM MQ Micro Focus COBOL de 32 bits existentes utilizando `mqmcb.lib` o `mqiccb.lib`, en lugar de las bibliotecas `mqmcbb` y `mqiccbb` .

Para compilar, por ejemplo, el programa de ejemplo `amq0put0` utilizando Micro Focus COBOL:

1. Establezca la variable de entorno COBCPY para que haga referencia a los libros de copias de IBM MQ COBOL (especifique el código siguiente en una línea):

   ```
   set COBCPY= MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook
   ```

2. Compile el programa para obtener un archivo de objeto:

   ```
   cobol amq0put0 LITLINK
   ```

3. Enlace el archivo de objeto con el sistema de tiempo de ejecución.

- Establezca la variable de entorno LIB para que haga referencia a las bibliotecas COBOL del compilador.
- Enlace el archivo de objeto para su uso en el servidor IBM MQ:

```
cbllink amq0put0.obj mqmcb.lib
```

- O enlace el archivo de objeto para su uso en el cliente IBM MQ:

```
cbllink amq0put0.obj mqiccb.lib
```

Preparación de programas CICS y Transaction Server

Para compilar y enlazar un programa TXSeries para Windows NT V5.1 utilizando IBM VisualAge COBOL:

1. Establezca la variable de entorno (especifique el código siguiente en una línea):

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. Establezca la variable de entorno USERLIB:

```
set USERLIB=MQMCBB.LIB
```

3. Convierta, compile y enlace su programa:

```
cicstcl -l IBMCOB myprog.ccp
```

Esto se describe en la guía de *Programación de aplicaciones de Transaction Server para Windows NT V4*.

Para compilar y enlazar un programa CICS para Windows V5 utilizando Micro Focus COBOL:

- Establezca la variable INCLUDE:

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- Establezca la variable de entorno COBCPY:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- Establezca las opciones de COBOL:
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

y ejecute el código siguiente:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Preparación de programas Visual Basic en Windows*

Información a tener en cuenta cuando se utilizan programas de Microsoft Visual Basic en Windows.

Deprecated A partir de IBM MQ 9.0, el soporte para Microsoft Visual Basic 6.0 está en desuso. Las clases de IBM MQ para .NET son la tecnología de sustitución recomendada. Para obtener más información, consulte Desarrollo de aplicaciones .NET.

Nota: No se proporcionan las versiones de 64 bits de los archivos de módulo de Visual Basic.

Para preparar programas de Visual Basic en Windows:

1. Cree un proyecto nuevo.
2. Añada el archivo de módulo suministrado, CMQB.BAS, al proyecto.
3. Añada otros archivos de módulo proporcionados si los necesita:
   - CMQBB.BAS: soporte de MQAI
   - CMQCFB.BAS: soporte de PCF
   - CMQXB.BAS: soporte de salidas de canal
   - CMQPSB.BAS: publicación/suscripción

Consulte “Codificación en Visual Basic” en la página 1079 para obtener información sobre el uso de la llamada MQCONNXAny desde Visual Basic.

Llame al procedimiento MQ_SETDEFAULTS antes de realizar llamadas MQI en el código de proyecto. Este procedimiento configura las estructuras predeterminadas que requieren las llamadas MQI.

Especifique si está creando un servidor o un cliente de IBM MQ, antes de compilar o ejecutar el proyecto. Para ello, establezca la variable de compilación condicional *MqType*. Establezca *MqType* en un proyecto de Visual Basic en 1 para un servidor o en 2 para un cliente, como se muestra a continuación:

1. Seleccione el menú Proyecto.
2. Seleccione *Name* Propiedades (donde *Name* es el nombre del proyecto actual).
3. Seleccione la pestaña Crear en el recuadro de diálogo.
4. En el campo Argumentos de compilación condicional, especifique este valor para un servidor:

   ```
   MqType=1
   ```

   o este para un cliente:

   ```
   MqType=2
   ```

Conceptos relacionados

“Codificación en Visual Basic” en la página 1079
Información que se debe tener en cuenta al codificar programas de IBM MQ en Microsoft Visual Basic. Visual Basic solo está soportado en Windows.

Referencia relacionada

“Enlace de aplicaciones de Visual Basic con el código de IBM MQ MQI client” en la página 944
Puede enlazar aplicaciones de Microsoft Visual Basic con el código de IBM MQ MQI client en Windows.

Windows *Salida de seguridad SSPI*

IBM MQ for Windows proporciona una salida de seguridad para el IBM MQ MQI client y el servidor de IBM MQ. Se trata de un programa de salida de canal que proporciona autenticación para los canales de IBM MQ utilizando la interfaz de programación de servicios de seguridad (SSPI). La SSPI proporciona los recursos de seguridad integrados de los sistemas Windows.

Los paquetes de seguridad se cargan desde security.dll o secur32.dll. Estas DLL se suministran con el sistema operativo.

Se proporciona la autenticación unidireccional utilizando los servicios de autenticación NTLM. Se proporciona la autenticación bidireccional utilizando los servicios de autenticación Kerberos.

El programa de salida de seguridad se proporciona en formato fuente y de objeto. Puede utilizar el código de objeto tal como está, o puede utilizar el código fuente como punto de partida para crear sus propios programas de salida de usuario.

Consulte también "Utilización de la salida de seguridad SSPI en Windows" en la página 1164.

Introducción a las salidas de seguridad

Una salida de seguridad forma una conexión segura entre dos programas de salida de seguridad, uno se utiliza para el agente de canal de mensajes (MCA) de envío y el otro para el MCA de recepción.

El programa que inicia la conexión segura, es decir, el primer programa que toma el control después de establecer la sesión MCA, se conoce como *iniciador de contexto*. El programa interlocutor se conoce como *aceptador de contexto*.

En la tabla siguiente, se muestran algunos de los tipos de canal que son iniciadores de contexto y los aceptores de contexto asociados.

Tabla 149. Iniciadores de contexto y sus aceptores de contexto asociados

Iniciador de contexto	**Aceptador de contexto**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

El programa de salida de seguridad tiene dos puntos de entrada:

- **SCY_NTLM**

  Utiliza los servicios de autenticación NTLM, que proporcionan autenticación unidireccional. NTLM permite a los servidores verificar las identidades de sus clientes. No permite que los clientes verifiquen la identidad de un servidor o que un servidor verifique la identidad de otro. La autenticación NTLM se ha diseñado para un entorno de red donde se supone que los servidores son quienes dicen ser.

- **SCY_KERBEROS**

  Utiliza los servicios de autenticación mutua de Kerberos. El protocolo Kerberos no da por supuesto que los servidores de un entorno de red son genuinos. Las partes en ambos extremos de una conexión de red pueden verificar la identidad de la otra parte. Es decir, los servidores pueden verificar la identidad de los clientes y otros servidores, y los clientes pueden verificar la identidad de un servidor.

Qué hace la salida de seguridad

En este tema, se describe lo que hacen los programas de salida de canal SSPI.

Los programas de salida de canal suministrados proporcionan la autenticación unidireccional o bidireccional (mutua) de un sistema asociado cuando se está estableciendo una sesión. Para un determinado canal, cada programa de salida tiene un *principal* asociado (similar a un ID de usuario, consulte "Control de accesos de IBM MQ y principales de Windows" en la página 1048). Una conexión entre dos programas de salida es una asociación entre los dos principales.

Después de establecer la sesión subyacente, se establece una conexión segura entre dos programas de salida de seguridad (uno para el MCA emisor y otro para el MCA receptor). La secuencia de operaciones es la siguiente:

1. Cada programa está asociado con un determinado principal, por ejemplo, como resultado de una operación de inicio de sesión explícito.

2. El iniciador de contexto solicita una conexión segura con el socio del paquete de seguridad (para Kerberos, el socio indicado) y recibe una señal (denominada token1). La señal se envía al programa asociado, utilizando la sesión subyacente que ya se ha establecido.
3. El programa asociado (el aceptador de contexto) pasa token1 al paquete de seguridad, que verifica que el iniciador de contexto sea auténtico. Para NTLM, ahora se establece la conexión.
4. Para la salida de seguridad proporcionada por Kerberos (es decir, para la autenticación mutua), el paquete de seguridad también genera una segunda señal (llamada token2), que el aceptador de contexto devuelve al iniciador de contexto utilizando la sesión subyacente.
5. El iniciador de contexto utiliza token2 para verificar que el aceptador de contexto es auténtico.
6. En este momento, si ambas aplicaciones están satisfechas con la autenticidad de la señal del socio, se establece la conexión segura (autenticada).

Control de accesos de IBM MQ y principales de Windows

El control de accesos que proporciona IBM MQ se basa en el usuario y el grupo. La autenticación que proporciona Windows se basa en principales, por ejemplo, el usuario y el valor de servicePrincipalName (SPN). En el caso de servicePrincipalName, es posible que un solo usuario tenga varios asociados.

La salida de seguridad SSPI utiliza los principales de Windows relevantes para la autenticación. Si la autenticación de Windows es satisfactoria, la salida pasa el ID de usuario que está asociado con el principal de Windows a IBM MQ para el control de accesos.

Los principales de Windows que son relevantes para la autenticación varían, en función del tipo de autenticación utilizado.

- Para la autenticación NTLM, el principal de Windows para el iniciador de contexto es el ID de usuario asociado al proceso que se está ejecutando. Como esta autenticación es unidireccional, el principal asociado con el aceptador de contexto es irrelevante.
- Para la autenticación Kerberos, en los canales CLNTCONN, el principal de Windows es el ID de usuario asociado con el proceso que se está ejecutando. De lo contrario, el principal de Windows es el servicePrincipalName que se forma añadiendo el siguiente prefijo a QueueManagerName.

```
ibmMQSeries/
```

z/OS Creación de una aplicación procedimental en z/OS

Las publicaciones de CICS, IMS y de z/OS describen cómo crear aplicaciones que se ejecutan en estos entornos.

Este grupo de temas describe las tareas adicionales y los cambios en las tareas estándar que debe realizar cuando crea aplicaciones IBM MQ for z/OS para estos entornos. Están soportados los lenguajes de programación COBOL, C, C++, Assembler y PL/I. Para obtener información acerca de cómo crear aplicaciones C++, consulte la sección Utilización de C++.

Las tareas que debe realizar para crear una aplicación IBM MQ for z/OS ejecutable dependen del lenguaje de programación en que se escribe el programa y del entorno en que se ejecutará la aplicación.

Además de codificar las llamadas MQI de su programa, añada las sentencias del lenguaje de adecuado para incluir el archivo de definición de datos IBM MQ for z/OS para el lenguaje que esté utilizando. Familiarícese con el contenido de estos archivos. Consulte “Archivos de definición de datos de IBM MQ” en la página 736 para obtener una descripción completa.

Nota

El nombre **thlqual** es el cualificador de alto nivel de la biblioteca de instalación en z/OS.

z/OS *Preparación del programa para su ejecución*

Después de haber escrito el programa para que la aplicación de IBM MQ cree una aplicación ejecutable, debe compilarlo o ensamblarlo y, a continuación, editar mediante enlaces el código de objeto resultante con el programa de apéndice que proporciona IBM MQ for z/OS para cada entorno al que da soporte.

La forma de preparar el programa depende tanto del entorno (por lotes, CICS, IMS(BMP o MPP), Linux o z/OS UNIX System Services) en el que se ejecuta la aplicación, como de la estructura de los conjuntos de datos de la instalación de z/OS .

En "Llamada dinámica al apéndice IBM MQ" en la página 1055, se describe un método alternativo para realizar llamadas MQI en los programas, para que no tenga que editar mediante enlaces un apéndice de IBM MQ. Este método no está disponible para todos los lenguajes y entornos.

No edite mediante enlaces un nivel superior del programa de apéndice que el de la versión de IBM MQ for z/OS en la que se ejecuta el programa. Por ejemplo, un programa que se ejecuta en MQSeries para OS/390, V5.2 no debe editarse mediante enlaces con un programa de apéndice suministrado con IBM MQ for z/OS V7.

z/OS *Compilación de una aplicación C de 64 bits*

En z/OS, las aplicaciones en C de 64 bits se compilan utilizando el compilador LP64 y las opciones del enlazador. El archivo de cabecera IBM MQ for z/OS *cmqc.h* reconoce cuándo se proporciona esta opción al compilador y genera tipos de datos y estructuras de IBM MQ adecuados para la operación de 64 bits.

Hay que compilar el código C con esta opción para utilizar las bibliotecas de enlace dinámico (DLL) adecuadas para la semántica de coordinación necesaria. Para ello, vincule el código compilado con la unidad lateral adecuada definida en la tabla siguiente:

Tabla 150. Nombre de la unidad lateral requerida en cada semántica de coordinación

Coordinación	**Nombre de la unidad lateral**
MQI de confirmación en fase única	CSQBMQ2X
Confirmación en dos fases con coordinación RRS, utilizando verbos RRS	CSQBRR2X
Confirmación en dos fases con coordinación RRS, utilizando verbos MQI	CSQBRI2X

Nota: Para las aplicaciones C de 31 bits, también puede establecer opciones de compilador para la interfaz de llamada (Language Environment o XPLINK), tal como se describe en "Creación de aplicaciones por lotes z/OS utilizando Language Environment o XPLINK de 31 bits" en la página 1051. Para aplicaciones C de 64 bits, no especifique la interfaz de llamada, porque el único enlace soportado es XPLINK.

Utilice el procedimiento JCL EDCQCB, proporcionado con *z/OS XL C/C++*, para crear un programa IBM MQ de confirmación en fase única como un trabajo por lotes como se indica a continuación:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',            < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',            < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//        DD
//        DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

Para crear un programa coordinado RRS en z/OS UNIX System Services, compile y enlace como se indica a continuación:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Creación de aplicaciones por lotes de z/OS*

Información sobre cómo crear aplicaciones por lotes de z/OS y los pasos que deben seguirse.

Para crear una aplicación para IBM MQ for z/OS que se ejecuta en z/OS por lotes, cree el lenguaje de control de trabajos (JCL) que realiza estas tareas:

1. Compile (o ensamble) el programa para generar el código de objeto. El JCL de la compilación debe incluir sentencias SYSLIB para que los archivos de definición de datos del producto estén disponibles para el compilador. Las definiciones de datos se proporcionan en las siguientes bibliotecas de IBM MQ for z/OS:
   - Para COBOL, **thlqual**.SCSQCOBC
   - Para el lenguaje ensamblador, **thlqual**.SCSQMACS
   - Para C, **thlqual**.SCSQC370
   - Para PL/I, **thlqual**.SCSQPLIC
2. Para una aplicación C, establezca un enlace previo con el código de objeto creado en el paso “1” en la página 1050.
3. Para las aplicaciones PL/I, utilice la opción de compilador EXTRN(SHORT).
4. Edite mediante enlaces el código de objeto creado en el paso “1” en la página 1050 (o el paso “2” en la página 1050 para una aplicación C) para producir un módulo de carga. Cuando edita mediante enlaces el código, debe incluir uno de los programas de apéndice por lotes de IBM MQ for z/OS (CSQBSTUB o uno de los programas de apéndice RRS: CSQBRRSI o CSQBRSTB).

   CSQBSTUB
   confirmación en una sola fase proporcionada por IBM MQ for z/OS

   CSQBRRSI
   confirmación de dos fases proporcionada por RRS utilizando MQI

   CSQBRSTB
   confirmación de dos fases proporcionada por RRS directamente

   Notas:

   a. Si utiliza CSQBRSTB, también debe editar mediante enlaces la aplicación con ATRSCSS desde SYS1.CSSLIB. La Figura 113 en la página 1050 y la Figura 114 en la página 1051 muestran fragmentos de JCL sobre cómo hacerlo. Los apéndices son independiente del lenguaje y se proporcionan en la biblioteca **thlqual**.SCSQLOAD.

   b. Si la aplicación se ejecuta en Language Environment, debe asegurarse de que edita los enlaces con la DLL de Language Environment en su lugar, tal como se describe en “Creación de aplicaciones por lotes z/OS utilizando Language Environment o XPLINK de 31 bits” en la página 1051.
5. Almacene el módulo de carga en una biblioteca de carga de aplicaciones.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Figura 113. Fragmentos de JCL para editar mediante enlaces el módulo de objeto en el entorno de proceso por lotes utilizando la confirmación en una sola fase

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

Figura 114. Fragmentos de JCL para editar mediante enlaces el módulo de objeto en el entorno de proceso por lotes utilizando la confirmación en dos fases

Para ejecutar un programa de proceso por lotes o RRS, debe incluir las bibliotecas **thlqual**.SCSQAUTH y **thlqual**.SCSQLOAD en la concatenación de conjuntos de datos STEPLIB o JOBLIB.

Para ejecutar un programa TSO, debe incluir las bibliotecas **thlqual**.SCSQAUTH y **thlqual**.SCSQLOAD en la STEPLIB utilizada por la sesión TSO.

Para ejecutar un programa por lotes desde el shell z/OS UNIX System Services , añada las bibliotecas **thlqual**.SCSQAUTH y **thlqual**.SCSQLOAD a la especificación STEPLIB en $HOME? .profile de la forma siguiente:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Creación de aplicaciones por lotes z/OS utilizando Language Environment o XPLINK de 31 bits*

IBM MQ for z/OS proporciona un conjunto de bibliotecas de enlace dinámico (DLL) que se debe utilizar al editar los enlaces de las aplicaciones.

Existen dos variantes de las bibliotecas que permiten a la aplicación utilizar una de las siguientes interfaces de llamada:

- La interfaz de llamada de Language Environment de 31 bits.
- La interfaz de llamada XPLINK de 31 bits. z/OS XPLINK es un convenio de llamada de alto rendimiento disponible para aplicaciones en C. Consulte XPLINK | NOXPLINK en la documentación de z/OS 2.2 .

Para utilizar las DLL, la aplicación se vincula o enlaza con las llamadas *unidades laterales* (sidedecks), en lugar de los apéndices (stub) proporcionados con las versiones anteriores. Las unidades laterales se encuentran en la biblioteca SCSQDEFS (en lugar de en la biblioteca SCSQLOAD).

Tabla 151. Variantes de bibliotecas de enlace dinámico

Confirmar	**DLL de Language Environment de 31 bits**	**DLL de XPLINK de 31 bits**	**Nombre de apéndice equivalente**
Bibliotecas MQI de compromiso en una fase	CSQBMQ1	CSQBMQ1X	CSQBSTUB
Confirmación en dos fases con coordinación RRS utilizando verbos de control de transacción RRS	CSQBRR1	CSQBRR1X	CSQBRSTB

Tabla 151. Variantes de bibliotecas de enlace dinámico (continuación)			
Confirmar	**DLL de Language Environment de 31 bits**	**DLL de XPLINK de 31 bits**	**Nombre de apéndice equivalente**
Confirmación en dos fases con coordinación RRS utilizando verbos de control de transacción MQI	CSQBRI1	CSQBRI1X	CSQBRRSI

Nota: Todas las unidades laterales contienen una definición del punto de entrada de conversión de datos, MQXCNVC, resuelto previamente mediante la inclusión de CSQASTUB.

Problemas comunes:

- El siguiente mensaje aparece en el registro de trabajo si la aplicación utiliza consumos de mensajes asíncronos (llamadas MQCB, MQCTL o MQSUB) y no se utiliza la interfaz DLL anterior:

  ```
  CSQB001E Los programas de entorno de lenguaje que se ejecutan en z/OS por lotes o z/OS UNIX
  System Services deben utilizar la interfaz DLL para IBM MQ
  ```

  Solución: Vuelva a compilar la aplicación para que use unidades laterales en lugar de apéndices, tal y como se ha detallado anteriormente.

- En tiempo de compilación del programa, aparece este mensaje:

  ```
  IEW2469E Los atributos de una referencia a MQAPI-NAME de la sección your-code no coinciden con
  los atributos de
  el símbolo de destino
  ```

  Razón: Esto significa que se ha compilado el programa XPLINK con la versión V701 (o posterior) de cmqc.h, pero que se ha enlazado con unidades laterales.

  Solución: Cambie el archivo de compilación del programa para que se enlace con la correspondiente unidad lateral de SCSQDEFS en lugar de un apéndice de SCSQLOAD

El ejemplo de JCL de ejemplo ilustra cómo se puede compilar y editar enlaces de un programa C para utilizar la interfaz de llamada DLL de Language Environment de 31 bits:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Nota: La compilación utiliza la opción **DLL**. La edición de enlaces usa la opción **DYNAM=DLL** y las referencias a la biblioteca **CSQBMQ1**.

El ejemplo de JCL de ejemplo ilustra cómo se puede compilar y editar enlaces de un programa C para utilizar la interfaz de llamada DLL de XPLINK de 31 bits:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
```

```
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

Nota: La compilación utiliza las opciones **XPLINK** y **DLL**. La edición de enlaces usa la opción **DYNAM=DLL** y referencia la biblioteca **CSQBMQ1X**.

Asegúrese de añadir la opción de compilación DLL a cada programa del módulo. Mensajes como, por ejemplo, IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED indican que hay que comprobar que todos los programas se han compilado con la opción DLL.

z/OS *Creación de aplicaciones CICS en z/OS*

Utilice esta información cuando cree aplicaciones CICS en z/OS.

Para crear una aplicación para IBM MQ for z/OS que se ejecuta en CICS, debe:

- Convertir los mandatos de CICS del programa al lenguaje en el que está escrito el resto del programa.
- Compilar o ensamblar la salida del conversor para generar el código de objeto.
  - Para los programas PL/I, utilice la opción de compilador EXTRN(SHORT).
  - Para aplicaciones C, si la aplicación no utiliza XPLINK, utilice la opción de compilador DEFINE (MQ_OS_LINKAGE=1).
- Editar mediante enlaces el código de objeto para crear un módulo de carga.

CICS proporciona un procedimiento para ejecutar estos pasos en orden para cada uno de los lenguajes de programación a los que da soporte.

- Para CICS Transaction Server para z/OS, la publicación *CICS Transaction Server for z/OS System Definition Guide* describe cómo utilizar estos procedimientos y la publicación *CICS/ESA Application Programming Guide* incluye más información sobre el proceso de conversión.

Debe incluir:

- En la sentencia SYSLIB de la etapa de compilación (o ensamblaje), las sentencias para que los archivos de definición de datos del producto estén disponibles para el compilador. Las definiciones de datos se proporcionan en las siguientes bibliotecas de IBM MQ for z/OS:
  - Para COBOL, **thlqual**.SCSQCOBC
  - Para el lenguaje ensamblador, **thlqual**.SCSQMACS
  - Para C, **thlqual**.SCSQC370
  - Para PL/I, **thlqual**.SCSQPLIC
- En el JCL de edición mediante enlaces, el programa de apéndice IBM MQ for z/OS CICS (CSQCSTUB). La Figura 115 en la página 1054 muestra fragmentos del código JCL sobre cómo hacerlo. El apéndice es independiente del lenguaje y se proporciona en la biblioteca **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figura 115. Fragmentos de JCL para editar mediante enlaces el módulo de objeto en el entorno CICS

- Para las versiones de CICS anteriores a CICS TS 3.2, o si desea utilizar las API de propiedad de mensaje de IBM MQ o las API de IBM MQ MQCB, MQCTL, MQSTAT, MQSUB o MQSUBR, debe enlazar mediante enlaces el código de objeto con el apéndice suministrado de CICS, DFHMQSTB, no el CSQCSTUB suministrado de IBM MQ. Para obtener más información sobre cómo crear programas IBM MQ para CICS, consulte Programa de apéndice de API para acceder a las llamadas MQI de IBM MQ en la documentación del producto CICS.

Cuando haya realizado estos pasos, almacene el módulo de carga en una biblioteca de carga de aplicaciones y defina el programa en CICS de la forma habitual.

Para poder ejecutar un programa CICS, el administrador del sistema debe definirlo previamente en CICS como un programa y una transacción de IBM MQ. A continuación, puede ejecutarlo de la forma habitual.

z/OS *Creación de aplicaciones IMS (BMP o MPP)*

Utilice esta información al crear aplicaciones IMS (BMP o MPP).

Si está creando programas DL/I de proceso por lotes, consulte “Creación de aplicaciones por lotes de z/OS” en la página 1050. Para crear otras aplicaciones que se ejecutan en IMS (ya sea como BMP o MPP), cree un JCL que realice estas tareas:

1. Compile (o ensamble) el programa para generar el código de objeto. El JCL de la compilación debe incluir sentencias SYSLIB para que los archivos de definición de datos del producto estén disponibles para el compilador. Las definiciones de datos se proporcionan en las siguientes bibliotecas de IBM MQ for z/OS:
   - Para COBOL, **thlqual**.SCSQCOBC
   - Para el lenguaje ensamblador, **thlqual**.SCSQMACS
   - Para C, **thlqual**.SCSQC370
   - Para PL/I, **thlqual**.SCSQPLIC
2. Para una aplicación C, establezca un enlace previo con el módulo de objeto creado en el paso “1” en la página 1054.
3. Para los programas PL/I, utilice la opción de compilador EXTRN(SHORT).
4. Para una aplicación C, si la aplicación no utiliza XPLINK, utilice la opción de compilador DEFINE (MQ_OS_LINKAGE=1).
5. Edite mediante enlaces el código de objeto creado en el paso “1” en la página 1054 (o el paso “2” en la página 1054 para una aplicación C/370) para producir un módulo de carga:
   a. Incluya el módulo de interfaz de lenguaje de IMS (DFSLI000).
   b. Incluya el programa de apéndice de IBM MQ for z/OS IMS stub program (CSQQSTUB). La Figura 116 en la página 1055 muestra fragmentos de JCL sobre cómo hacerlo. El apéndice es independiente del lenguaje y se proporciona en la biblioteca **thlqual**.SCSQLOAD.

      Nota: Si utiliza COBOL, seleccione la opción de compilador NODYNAM para que el editor de enlace pueda resolver las referencias a CSQQSTUB, a menos que tenga previsto utilizar un enlace dinámico como se describe en “Llamada dinámica al apéndice IBM MQ” en la página 1055.
6. Almacene el módulo de carga en una biblioteca de carga de aplicaciones.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figura 116. Fragmentos de JCL para editar mediante enlaces el módulo de objeto en el entorno IMS

Para poder ejecutar un programa IMS, el administrador del sistema debe definirlo previamente en IMS como un programa y una transacción de IBM MQ. A continuación, puede ejecutarlo de la forma habitual.

z/OS *Compilación de aplicaciones de z/OS UNIX System Services*

Utilice esta información al crear aplicaciones z/OS UNIX System Services .

Para crear una aplicación C para IBM MQ for z/OS que se ejecuta bajo z/OS UNIX System Services, compile y enlace la aplicación de la forma siguiente:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

donde **thlqual** es el calificador de alto nivel utilizado por la instalación.

Para ejecutar el programa en C, hay que añadir lo siguiente al archivo `.profile`; esto tiene que estar en el directorio raíz:

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

Tenga en cuenta que es necesario salir de z/OS UNIX System Servicesy volver a especificar z/OS UNIX System Services para que se reconozca el cambio.

Si desea ejecutar varios shells, añada la palabra export al principio de la línea, es decir:

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

Una vez que haya esto correctamente, podrá enlazar CSQBSTUB y emitir llamadas IBM MQ.

En “Llamada dinámica al apéndice IBM MQ” en la página 1055, se describe un método alternativo para realizar llamadas MQI en los programas, para que no tenga que editar mediante enlaces un apéndice de IBM MQ. Este método no está disponible para todos los lenguajes y entornos.

No edite mediante enlaces un nivel superior del programa de apéndice que el de la versión de IBM MQ for z/OS en la que se ejecuta el programa. Por ejemplo, un programa que se ejecuta en IBM WebSphere MQ for z/OS 7.1 no se debe enlazar con un programa de apéndice proporcionado con IBM MQ for z/OS 8.0.

z/OS *Llamada dinámica al apéndice IBM MQ*

En lugar de editar el enlace del programa apéndice de IBM MQ con el código de objeto, puede invocar dinámicamente el apéndice desde dentro de su programa.

Puede hacer esto en los entornos de proceso por lotes, IMS y CICS. Este recurso no está soportado en el entorno RRS. Si el programa de aplicación utiliza RRS para coordinar actualizaciones, consulte “Consideraciones RRS” en la página 1060.

Sin embargo, este método:

- Aumenta la complejidad de los programas
- Aumenta el almacenamiento necesario para los programas en tiempo de ejecución
- Reduce el rendimiento de los programas

- Significa que no puede utilizar los mismos programas en otros entornos

Si llama al apéndice dinámicamente, el programa apéndice adecuado y sus alias deben estar disponibles en tiempo de ejecución. Para garantizar esto, incluya el conjunto de datos de IBM MQ for z/OS SCSQLOAD:

- Para el proceso por lotes y IMS, en la concatenación STEPLIB del JCL.
- Para CICS, en la concatenación CICS DFHRPL.

Para IMS, asegúrese de que la biblioteca que contiene el apéndice dinámico (creada tal como se describe en la información sobre cómo instalar el adaptador IMS en Configuración del adaptador IMS) está por delante del conjunto de datos SCSQLOAD en la concatenación STEPLIB del JCL de la región.

Utilice los nombres que se muestran en Tabla 152 en la página 1056 cuando llama al apéndice dinámicamente. En PL/I, declare únicamente los nombres de llamada utilizados en el programa.

Tabla 152. Nombres de llamada para el enlace dinámico

Llamada MQI	Nombres de llamadas dinámicas por lotes (no RRS)	Nombres de llamadas dinámicas de CICS	Nombres de llamadas dinámicas de IMS
MQBACK	CSQBBACK	sin soporte	No soportado
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	No soportado
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	sin soporte	No soportado
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	No soportado
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGet	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Nota: 1. Estas llamadas de API sólo están disponibles cuando se utiliza CICS TS 3.2 o posterior y se debe utilizar el CSQCSTUB que se entrega con CICS. Para CICS TS 3.2, debe estar aplicado el APAR PK66866. Para CICS TS 4.1, debe estar aplicado el APAR PK89844.

Para obtener ejemplos de cómo utilizar esta técnica, consulte las figuras siguientes:

- Lote y COBOL: consulte Figura 117 en la página 1057
- CICS y COBOL: consulte Figura 118 en la página 1057
- IMS y COBOL: consulte Figura 119 en la página 1058
- Lote y ensamblador: consulte Figura 120 en la página 1058
- CICS y ensamblador: consulte Figura 121 en la página 1058
- IMS y ensamblador: consulte Figura 122 en la página 1058
- Lote y C: Figura 123 en la página 1059
- CICS y C: consulte Figura 124 en la página 1059
- IMS y C: consulte Figura 125 en la página 1059
- Lote y PL/I: consulte Figura 126 en la página 1059
- IMS y PL/I: consulte Figura 127 en la página 1060

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figura 117. Enlace dinámico utilizando COBOL en el entorno de proceso por lotes

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figura 118. Enlace dinámico utilizando COBOL en el entorno de CICS

```
...
      WORKING-STORAGE SECTION.
...
         05 WS-MQOPEN                    PIC X(8) VALUE 'MQOPEN'.
...
      PROCEDURE DIVISION.
...
          CALL WS-MQOPEN WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
     * ------------------------------------------------------------ *
     *
     *    If the compilation option 'DYNAM' is specified
     *    then you may code the MQ calls as follows
     *
     * ------------------------------------------------------------ *
...
          CALL 'MQOPEN'  WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
```

Figura 119. Enlace dinámico utilizando COBOL en el entorno de IMS

```
...
        LOAD   EP=CSQBOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=CSQBOPEN
...
```

Figura 120. Enlace dinámico utilizando el lenguaje ensamblador del entorno de proceso por lotes

```
...
        EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figura 121. Enlace dinámico utilizando el lenguaje ensamblador del entorno CICS

```
...
        LOAD   EP=MQOPEN
...
        CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
        DELETE EP=MQOPEN
...
```

Figura 122. Enlace dinámico utilizando el lenguaje ensamblador del entorno IMS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 123. Enlace dinámico utilizando el lenguaje C del entorno de proceso por lotes

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 124. Enlace dinámico utilizando el lenguaje C del entorno CICS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 125. Enlace dinámico utilizando el lenguaje C del entorno IMS

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figura 126. Enlace dinámico utilizando PL/I en el entorno de proceso por lotes

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figura 127. Enlace dinámico utilizando PL/I en el entorno de IMS

z/OS *Consideraciones RRS*

Tenga en cuenta esta información si su programa de aplicación utiliza RRS para coordinar las actualizaciones.

IBM MQ proporciona dos apéndices distintos para programas por lotes que necesiten coordinación RSS; consultar "El adaptador de proceso por lotes RRS" en la página 914. La diferencia en el comportamiento de las llamadas de API posteriores se determina en el momento de MQCONN por el adaptador por lotes a partir de la información pasada por la rutina de apéndices en la API MQCONN o MQCONNX. Esto significa que las llamadas de API dinámicas están disponibles para los programas por lotes que necesitan la coordinación de RRS, siempre que la conexión inicial a IBM MQ se haya realizado utilizando el apéndice adecuado. En el ejemplo siguiente se ilustra esto:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS *Depurar los programas*

Utilice esta información para depurar programas TSO y CICS, y para aprender más sobre el rastreo de CICS.

Las ayudas principales para depurar programas de aplicación de IBM MQ for z/OS son los códigos de razón devueltos por cada llamada de API. Para obtener un listado, incluidas ideas para la acción correctiva, consulte:

- IBM MQ for z/OS mensajes, finalización, y códigos de razón para IBM MQ for z/OS
- Mensajes y códigos de razón para todas las demás plataformas IBM MQ

En este tema también se sugieren otras herramientas de depuración que se deben utilizar en entornos concretos.

Depuración de programas TSO

Las siguientes herramientas de depuración interactiva están disponibles para los programas TSO:

- herramienta TEST
- herramienta de depuración interactiva de VS COBOL II
- herramienta de depuración interactiva INSPECT para programas C y PL/I

Depuración de programas CICS

Puede utilizar el Execution Diagnostic Facility (CEDF) de CICS para probar los programas de CICS interactivamente sin tener que modificar el procedimiento del programa o de preparación del programa.

Para obtener más información acerca de EDF, consulte la publicación *CICS Transaction Server for z/OS CICS Application Programming Guide*.

Rastreo de CICS

Probablemente también le resultará útil utilizar la transacción de control de rastreo (CETR) de CICS para controlar la actividad de rastreo de CICS.

Para obtener más información acerca de CETR, consulte el manual *CICS Transaction Server for z/OS CICS-Supplied Transactions*.

Para determinar si el rastreo de CICS está activo, visualice el estado de conexión utilizando el panel CKQC. Este panel también muestra el número de rastreo.

Para interpretar las entradas de rastreo de CICS, consulte Tabla 153 en la página 1061.

La entrada de rastreo de CICS para estos valores es AP0 *xxx* (donde *xxx* es el número de rastreo especificado cuando se ha habilitado el adaptador CICS). Todas las entradas de rastreo salvo CSQCTEST se emiten mediante CSQCTRUE. CSQCTEST se emite mediante CSQCRST y CSQCDSP.

Tabla 153. Entradas de rastreo de adaptador de CICS

Nombre	Descripción	Secuencia de rastreo	Datos de rastreo
CSQCABNT	Terminación anómala	Antes de emitir END_THREAD ABNORMAL a IBM MQ. Esto se debe al final de la tarea y la aplicación podría llevar a cabo una restitución implícita. En este caso, se incluye una solicitud ROLLBACK en la llamada END_THREAD.	Información de unidad de trabajo. Puede utilizar esta información cuando encuentre información sobre el estado de trabajo. (Por ejemplo, se puede verificar con la salida producida por el mandato DISPLAY THREAD o el programa de utilidad de impresión de registro IBM MQ for z/OS.)
CSQCBACK	Restitución de punto de sincronización	Antes de emitir BACKOUT a IBM MQ for z/OS. Esto se debe a una solicitud de restitución explícita de la aplicación.	Información de unidad de trabajo.
CSQCCCRC	Código de terminación y código de razón	Después de un retorno no satisfactorio de la llamada de API.	Código de terminación y código de razón.

Tabla 153. Entradas de rastreo de adaptador de CICS (continuación)

Nombre	Descripción	Secuencia de rastreo	Datos de rastreo
CSQCCOMM	Confirmación de punto de sincronización	Antes de emitir COMMIT a IBM MQ for z/OS. Esto se puede deber a una solicitud de confirmación de una sola fase o a la segunda fase de una solicitud de confirmación de dos fases. La solicitud se debe a una solicitud de punto de sincronización explícita de la aplicación.	Información de unidad de trabajo.
CSQCEXER	Ejecutar resolución	Antes de emitir EXECUTE_RESOLVE a IBM MQ for z/OS.	La información de unidad de trabajo de la unidad de trabajo que emite EXECUTE_RESOLVE. Esta es la última unidad de trabajo en duda en el proceso de resincronización.
CSQCGETW	Espera GET	Antes de emitir la espera de CICS.	Dirección del ECB al que esperar.
CSQCGMGD	Datos de mensaje GET	Después de un retorno satisfactorio de MQGET.	Hasta 40 bytes de los datos del mensaje.
CSQCGMGH	Manejador de mensajes GET	Antes de emitir MQGET a IBM MQ for z/OS.	El manejador del objeto.
CSQCGMGI	Obtener ID del mensaje	Después de un retorno satisfactorio de MQGET.	ID de mensaje e ID de correlación del mensaje.
CSQCINDL	Lista de dudas	Después de un retorno satisfactorio de la segunda INQUIRE_INDOUBT.	Las unidades dudosas de la lista de trabajo.
CSQCINDO	Sólo para uso de IBM		
CSQCINDS	Tamaño de la lista de dudosos	Después de un retorno satisfactorio desde la primera INQUIRE_INDOUBT y la lista de dudosos no está vacía.	Longitud de la lista. Dividido por 64 da el número de unidades de trabajo dudosas.
CSQCINQH	Manejador INQ	Antes de emitir MQINQ a IBM MQ for z/OS.	El manejador del objeto.
CSQCLOSH	Manejador CLOSE	Antes de emitir MQCLOSE a IBM MQ for z/OS.	El manejador del objeto.
CSQCLOST	Pérdida de disposición	Durante el proceso de resincronización, CICS informa al adaptador de que se ha reiniciado, de modo que no hay disponible información de eliminación relativa a la unidad de trabajo que se está resincronizando.	ID de unidad de trabajo conocido por CICS para la unidad de trabajo que se está resincronizando.

Tabla 153. Entradas de rastreo de adaptador de CICS (continuación)

Nombre	Descripción	Secuencia de rastreo	Datos de rastreo
CSQCNIND	Disposición no dudosa	Durante el proceso de resincronización, CICS informa al adaptador de que la unidad de trabajo que se está resincronizando no debería haber estado en duda (es decir, tal vez todavía se esté ejecutando).	ID de unidad de trabajo conocido por CICS para la unidad de trabajo que se está resincronizando.
CSQCNORT	Terminación normal	Antes de emitir END_THREAD NORMAL a IBM MQ for z/OS. Esto se hace al final de la tarea y, por lo tanto, la aplicación puede llevar a cabo una confirmación de punto de sincronización implícita. En este caso, se incluye una solicitud COMMIT en la llamada END_THREAD.	Información de unidad de trabajo.
CSQCOPNH	Manejador OPEN	Después de un retorno satisfactorio de MQOPEN.	El manejador del objeto.
CSQCOPNO	Objeto OPEN	Antes de emitir MQOPEN a IBM MQ for z/OS.	El nombre del objeto.
CSQCPMGD	Datos de mensaje PUT	Antes de emitir MQPUT a IBM MQ for z/OS.	Hasta 40 bytes de los datos del mensaje.
CSQCPMGH	Manejador de mensajes PUT	Antes de emitir MQPUT a IBM MQ for z/OS.	El manejador del objeto.
CSQCPMGI	ID de mensaje PUT	Tras un MQPUT correcto desde IBM MQ for z/OS.	ID de mensaje e ID de correlación del mensaje.
CSQCPREP	Preparación de punto de sincronización	Antes de emitir PREPARE para IBM MQ for z/OS en la primera fase del procesamiento de confirmación en dos fases. Esta llamada también se puede emitir desde el componente de gestión de colas distribuidas como una llamada de API.	Información de unidad de trabajo.
CSQCP1MD	Datos de mensaje PUTONE	Antes de emitir MQPUT1 a IBM MQ for z/OS.	Hasta 40 bytes de datos del mensaje.
CSQCP1MI	ID de mensaje PUTONE	Después de un retorno satisfactorio de MQPUT1.	ID de mensaje e ID de correlación del mensaje.
CSQCP1ON	Nombre de objeto PUTONE	Antes de emitir MQPUT1 a IBM MQ for z/OS.	El nombre del objeto.
CSQCRBAK	Restitución resuelta	Antes de emitir RESOLVE_ROLLBACK a IBM MQ for z/OS.	Información de unidad de trabajo.
CSQCRCMT	Confirmación resuelta	Antes de emitir RESOLVE_COMMIT a IBM MQ for z/OS.	Información de unidad de trabajo.

Tabla 153. Entradas de rastreo de adaptador de CICS (continuación)

Nombre	Descripción	Secuencia de rastreo	Datos de rastreo
CSQCRMIR	Respuesta RMI	Antes de volver a la RMI (interfaz del gestor de recursos) de CICS desde una invocación específica.	Valor de respuesta del RMI con estructura. Su significado depende del tipo de la invocación. Estos valores se documentan en la publicación *CICS Transaction Server for z/OS Customization Guide*. Para determinar el tipo de invocación, consulte las entradas de rastreo anteriores producidas por el componente RMI de CICS.
CSQCRSYN	Resincronización	Antes de que el proceso de resincronización inicia la tarea.	ID de unidad de trabajo conocido por CICS para la unidad de trabajo que se está resincronizando.
CSQCSETH	Manejador SET	Antes de emitir MQSET a IBM MQ for z/OS.	El manejador del objeto.
CSQCTASE	Sólo para uso de IBM		
CSQCTEST	Prueba de rastreo	Se utiliza en la llamada EXEC CICS ENTER TRACE para verificar el número de rastreo proporcionado por el usuario o el estado de rastreo de la conexión.	Sin datos.
CSQCDCFF	Sólo para uso de IBM		

Tratamiento de errores en un programa procedimental

Esta información explica los errores asociados a las llamadas MQI de una aplicación cuando se efectúa una llamada o cuando el mensaje se entrega en su destino final.

En la medida de lo posible, el gestor de colas devuelve todos los errores tan pronto como se realiza una llamada MQI. Estos son *errores determinados localmente*.

Cuando se envían mensajes a una cola remota, puede que haya errores no aparentes al efectuar una llamada MQI. En tal caso, el gestor de colas que identifica los errores, los notifica enviando otro mensaje al programa originador. Estos son *errores determinados remotamente*.

Errores determinados localmente

La información acerca de los errores determinados localmente incluye: error en una llamada MQI, interrupciones del sistema y mensajes que contienen datos incorrectos.

Las tres causas más comunes de los errores que puede notificar inmediatamente el gestor de colas son:

- Error de una llamada MQI debido, por ejemplo, a que una cola está llena
- Una interrupción de la ejecución de alguna parte del sistema de la que depende su aplicación, por ejemplo, el gestor de colas
- Los mensajes que contienen datos no se pueden procesar correctamente

Si está utilizando el recurso de colocación en cola asíncrona, los errores no se notifican de forma inmediata. Utilice la llamada MQSTAT para recuperar información de estado sobre las operaciones de transferencia asíncrona anteriores.

Error en una llamada MQI

El gestor de colas puede notificar cualquier error de codificación de una llamada MQI de forma inmediata. Esto lo lleva a cabo utilizando un conjunto de códigos de retorno predefinidos. Estos códigos están divididos en códigos de terminación y códigos de razón.

Para mostrar si una llamada se realiza correctamente, el gestor de colas devuelve un *código de terminación* cuando se completa la llamada. Hay tres códigos de terminación que indican que la llamada se ha realizado correctamente, ha terminado parcialmente o ha fallado. El gestor de colas también devuelve un *código de razón* que indica el motivo de la terminación parcial o del error de la llamada.

Los códigos de terminación y razón de cada llamada se listan, junto con la descripción de dicha llamada, en la sección Códigos de razón. Para obtener información detallada, junto con ideas para una acción correctiva, consulte:

- z/OS IBM MQ for z/OS mensajes, finalización, y códigos de razón para IBM MQ for z/OS
- Mensajes y códigos de razón para todas las demás plataformas IBM MQ

Diseñe sus programas para que manejen todos los códigos de retorno que pueda generar cada llamada.

Interrupciones de System i

Es posible que su aplicación desconozca cualquier interrupción si el gestor de colas al que está conectada se ha de recuperar de una anomalía del sistema. No obstante, debe diseñar su aplicación para asegurarse de que sus datos no se pierdan si se produce una interrupción de este tipo.

Los métodos que puede utilizar para asegurarse de que sus datos continúan siendo coherentes dependen de la plataforma en la que se ejecuta su gestor de colas:

z/OS z/OS

En los entornos de CICS y de IMS, puede realizar llamadas MQPUT y MQGET en unidades de trabajo gestionadas por CICS o IMS. En el entorno de procesos por lotes, puede realizar llamadas MQPUT y MQGET del mismo modo, pero debe asegurarse de que declara puntos de sincronización utilizando:

- Las llamadas MQCMIT y MQBACK de IBM MQ for z/OS (consulte “Confirmación y restitución de unidades de trabajo” en la página 874) o
- Los servicios RRS (Recoverable Resource Manager Services) de la gestión de transacciones de z/OS para proporcionar soporte de punto de sincronización de dos fases. RRS permite actualizar IBM MQ y otros recursos habilitados por RRS, tales como los recursos de procedimientos almacenados de Db2, en una única unidad de trabajo lógica. Para obtener información acerca del soporte del punto de sincronización RRS, consulte “Servicios de gestión de transacciones y gestor de recursos recuperables” en la página 879.

IBM i IBM i

Puede realizar sus llamadas MQPUT y MQGET en las unidades de trabajo globales gestionadas por el control de compromiso de IBM i. Puede declarar puntos de sincronización utilizando los mandatos COMMIT y ROLLBACK nativos de IBM i o los mandatos específicos del lenguaje. Las unidades de trabajo locales las gestiona IBM MQ utilizando las llamadas MQCMIT y MQBACK.

AIX, Linux, and Windows sistemas

En estos entornos, puede realizar sus llamadas MQPUT y MQGET de forma habitual, pero debe declarar puntos de sincronización utilizando las llamadas MQCMIT y MQBACK. Consulte “Confirmación y restitución de unidades de trabajo” en la página 874. En el entorno de CICS, los mandatos MQCMIT y MQBACK están inhabilitados, debido a que puede realizar sus llamadas MQPUT y MQGET en unidades de trabajo gestionadas por CICS.

Utilice los mensajes persistentes para transferir de datos cuya pérdida no se puede permitir. Se crean nuevas instancias de los mensajes persistentes si el gestor de colas necesita una recuperación después de una anomalía. ALW Con IBM MQ en AIX, Linux, and Windows, una llamada MQGET o MQPUT dentro de la aplicación fallará en el punto de llenar todos los archivos de registro, con el mensaje

MQRC_RESOURCE_PROBLEM. Para obtener información acerca de los archivos de registro en AIX, Linux, and Windows, consulte Administración IBM MQ. z/OS Para z/OS, consulte Planificación en z/OS.

Si un operador detiene el gestor de colas mientras se está ejecutando una aplicación, normalmente se utiliza la opción de desactivar temporalmente. El gestor entra en un estado de desactivación temporal en el que las aplicaciones pueden continuar funcionando, pero deben finalizar en cuanto resulte adecuado. Las aplicaciones rápidas y pequeñas pueden omitir el estado de desactivación temporal y pueden continuar hasta que finalicen con normalidad. Las aplicaciones de ejecución más larga, o las que esperan la llegada de mensajes, deben utilizar la opción *Error si está en fase de inmovilización* cuando utilizan las llamadas MQOPEN, MQPUT, MQPUT1 y MQGET. Estas opciones significan que cuando se desactiva temporalmente el gestor de colas las llamadas pero que la aplicación todavía tiene tiempo para realizar una finalización limpia emitiendo las llamadas que omiten el estado de desactivación temporal. Estas aplicaciones también puede confirmar o restituir los cambios que han realizado y, a continuación, finalizar.

Si se fuerza la detención del gestor de colas, esto es, se detiene sin desactivarlo temporalmente, las aplicaciones reciben el código de razón MQRC_CONNECTION_BROKEN cuando realizan llamadas MQI.

Salga de la aplicación o, de forma alternativa, en los sistemas IBM i IBM MQ for IBM i, AIX, Linux, and Windows emita una llamada MQDISC.

Mensajes que contienen datos incorrectos

Cuando utiliza unidades de trabajo en sus aplicaciones, si un programa no puede procesar correctamente un mensaje que recupera de una cola, se restituye una llamada MQGET.

El gestor de colas mantiene un recuento (en el campo *BackoutCount* del descriptor de mensajes) del número de veces que sucede. Este recuento lo mantiene en el descriptor de cada mensaje afectado. Este recuento puede proporcionar información importante acerca de la eficacia de una aplicación. Los mensajes cuyos recuentos de restitución que aumentan con el tiempo son mensajes que han sido rechazados de forma repetitiva. Diseñe su aplicación de modo que analice las razones y maneje estos mensajes como corresponda.

z/OS En IBM MQ for z/OS, para el recuero de restitución sobreviva los reinicios del gestor de colas, establezca el atributo **HardenGetBackout** en MQQA_BACKOUT_HARDENED, de lo contrario, si se ha de reiniciar el gestor de colas, no mantiene un recuento de restituciones para cada mensaje. Establecer el atributo de este modo aumenta el riesgo de procesos adicionales.

En sistemas IBM MQ para IBM i IBM i, AIX, Linux, and Windows , el recuento de restituciones siempre sobrevive a los reinicios del gestor de colas.

z/OS Asimismo, en IBM MQ for z/OS, cuando elimina mensajes de una cola de una unidad de trabajo, puede marcar un mensaje de modo que no vuelva a estar disponible si la aplicación vuelve a restituir la unidad de trabajo. Le mensaje marcado se trata como si se hubiera recuperado bajo una nueva unidad de trabajo. Marque el mensaje que ha de omitir la restitución utilizando la opción MQGMO_MARK_SKIP_BACKOUT.(en la estructura MQGMO) cuando utiliza la llamada MQGET. Consulte la sección “Omisión de restitución” en la página 820 para obtener más información acerca de esta técnica.

Cómo utilizar los mensajes de informes para la determinación de problemas

Cuando realiza su llamada MQI, el gestor de colas remoto no puede notificar errores, tales como un error de transferencia de un mensaje a una cola, pero puede enviarle un mensaje indicando cómo ha procesado su mensaje.

En la aplicación, puede crear mensajes de informe (MQPUT), así como seleccionar la opción para recibirlos y, en tal caso, los enviará otra aplicación o un gestor de colas.

Crear mensajes de informes

Los mensajes de informe permite que una aplicación indique a otra aplicación que no puede manejar el mensaje enviado.

No obstante, inicialmente se debe analizar el campo *Report* para determinar si la aplicación que ha enviado el mensaje está interesada en que se le notifique cualquier problema. Una vez se ha determinado que se requiere un mensaje de informe, tiene que decidir:

- Si desea incluir el mensaje original completo, simplemente los primeros 100 bytes de datos o nada del mensaje original.
- Qué se ha de hacer con el mensaje original. Puede descartarlo o dejar que vaya a la cola de mensajes no entregados.
- Si también es necesario el contenido de los campos *MsgId* y *CorrelId*.

Utilice el campo *Feedback* para indicar el motivo por el que se genera el mensaje de informe. Coloque sus mensajes de informe en la cola de respuesta de la aplicación. Consulte la sección Feedback para obtener más información.

Solicitar y recibir (MQGET) mensajes de informe

Cuando envía un mensaje a otra aplicación, si le notifica cualquier problema, a menos que rellene el campo *Report* para indicar el comentario que requiere. Consulte la sección Estructura del campo de informe para ver las opciones disponibles.

Los gestores de colas siempre colocan los mensajes de informe en una cola de respuesta y se le recomienda que sus aplicaciones hagan lo mismo. Cuando utiliza la función de mensajes de informe, especifique su cola de respuesta en el descriptor de mensaje de su mensaje. De lo contrario, la llamada MQPUT fallará.

Su aplicación debe contener procedimientos que supervisen su cola de respuestas y procesen cualquier mensaje que lleguen a la misma. Recuerde que un mensaje de informe puede contener el mensaje original completo, los primeros 100 bytes del mensaje original o ningún contenido del mensaje original.

El gestor de colas establece el campo *Feedback* del mensaje de informe para indicar la razón del error. Por ejemplo, no existe la cola de destino. Sus programas deben hacer lo mismo.

Para obtener más información acerca de los mensajes de informe, consulte la sección “Mensajes de informe” en la página 21.

Errores determinados de forma remota

Cuando envía mensajes a una cola remota, incluso cuando el gestor de colas ha procesado su llamada MQI sin encontrar ningún errores, otros factores pueden afectar el modo en que un gestor de colas remoto maneja su mensaje.

Por ejemplo, es posible que su cola de destino esté llena o que ni siquiera exista. Si su mensaje lo han de manejar otros gestores de colas intermedios en la ruta a la cola de destino, cualquiera de ellos puede encontrar un error.

Problemas de entrega de un mensaje

Cuando falla una llamada MQPUT, puede intentar volver a colocar el mensaje en la cola, devolver al remitente o colocarlo en la cola de mensajes no entregados.

Cada opción tiene sus méritos, pero es posible que no desee volver a colocar un mensaje si la causa por la que ha fallado MQPUT es debida a que la cola de destino estaba llena. En este caso, colocarlo en la cola de mensajes no entregados le permite entregarlo a la cola de destino correcta posteriormente.

Reintentar la entrega de mensajes

Antes de colocar un mensaje en una cola de mensajes no entregados, un gestor de colas remoto intenta volver a colocar el mensaje en la cola si se han establecido los atributos *MsgRetryCount* y

MsgRetryInterval para el canal, o si existe un programa de salida de reintento que pueda utilizar. El nombre de este programa está incluido en el atributo del campo *MsgRetryExitId*.

Si el campo *MsgRetryExitId* está en blanco, se utilizan los valores de los atributos *MsgRetryCount* y *MsgRetryInterval*.

Si campo *MsgRetryExitId* no está en blanco, se ejecuta el programa de salida con este nombre. Para obtener más información acerca de cómo utilizar sus propios programas de salida, consulte la sección "Programas de salida de canal para canales de mensajes" en la página 986.

Devolver mensaje a emisor

Puede devolver un mensaje al emisor solicitando que se genere un mensaje de informe que incluya el mensaje original completo.

Consulte la sección "Mensajes de informe" en la página 21 para obtener información detallada sobre las opciones de mensajes de informes.

Utilización de la cola de mensajes no entregados

Cuando un gestor de colas no puede entregar un mensaje, intenta colocarlo en la cola de mensajes no entregados. Esta cola se debe definir cuando se instala el gestor de colas.

Los programas pueden utilizar la cola de mensajes no entregados de la misma manera en que la utiliza el gestor de colas. Para determinar el nombre de la cola de mensajes no entregados, abra el objeto del gestor de colas (mediante la llamada MQOPEN) y consulte el atributo **DeadLetterQName** (mediante la llamada MQINQ).

Cuando el gestor de colas coloca un mensaje en esta cola, añade una cabecera al mensaje, cuyo formato se describe en la estructura de la cabecera de mensajes no entregados (MQDLH); consulte MQDLH - Cabecera de mensajes no entregados. Esta cabecera incluye el nombre de la cola de destino y la razón por la que el mensaje se ha colocado en la cola de mensajes no entregados. Esta cabecera se debe eliminar y el problema se debe corregir para poder colocar el mensaje en la cola prevista. Además, el gestor de colas cambia el campo *Format* del descriptor de mensaje (MQMD) para indicar que el mensaje contiene una estructura MQDLH.

Estructura MQDLH

Se recomienda que añada una estructura MQDLH a todos los mensajes que ponga en la cola de mensajes no entregados; pero si piensa utilizar el controlador de mensajes no entregados proporcionado por determinados productos IBM MQ, debe añadir una estructura MQDLH a los mensajes.

La adición de la cabecera a un mensaje puede hacer que el mensaje sea demasiado largo para la cola de mensajes no entregados, por lo que debe asegurarse de que los mensajes sean más cortos que el tamaño máximo permitido para la cola de mensajes no entregados y poder dar cabida como mínimo al tamaño indicado por la constante MQ_MSG_HEADER_LENGTH. El tamaño máximo de los mensajes permitidos en una cola está determinado por el valor del atributo **MaxMsgLength** de la cola. Para la cola de mensajes no entregados, asegúrese de que este atributo esté establecido en el valor máximo permitido por el gestor de colas. Si la aplicación no puede entregar un mensaje, y el mensaje es demasiado largo para colocarlo en la cola de mensajes no entregados, siga los consejos que se proporcionan en la descripción de la estructura MQDLH.

Asegúrese de que se supervise la cola de mensajes no entregados y de que se procesen los mensajes que llegan a ella. El controlador de la cola de mensajes no entregados se ejecuta como un programa de utilidad de proceso por lotes y se puede utilizar para realizar diversas acciones en los mensajes seleccionados de la cola de mensajes no entregados. Para obtener más detalles, consulte el tema "Proceso de cola de mensajes no entregados" en la página 1069.

Si la conversión de datos es necesaria, el gestor de colas convierte la información de cabecera cuando se utiliza la opción MQGMO_CONVERT en la llamada MQGET. Si el proceso que coloca el mensaje es un MCA, la cabecera va seguida de todo el texto del mensaje original.

Los mensajes colocados en la cola de mensajes no entregados pueden experimentar un truncamiento si son demasiado largos para esta cola. Una indicación posible de esta situación es que los mensajes de la

cola de mensajes no entregados tengan la misma longitud que el valor del atributo **MaxMsgLength** de la cola.

Proceso de cola de mensajes no entregados
Esta información contiene información de la interfaz de programación de uso general cuando se utiliza el proceso de cola de mensajes no entregados.

El proceso de cola de mensajes no entregados depende de los requisitos del sistema local, pero tenga en cuenta lo siguiente cuando elabore la especificación:

- El mensaje puede identificarse como que tiene una cabecera de cola de mensajes no entregados, porque el valor del campo de formato en MQMD es MQFMT_DEAD_LETTER_HEADER.
- En IBM MQ for z/OS utilizando CICS, si un MCA coloca este mensaje en la cola de mensajes no entregados, el campo *PutApplType* es MQAT_CICS y el campo *PutApplName* es el *ApplId* del sistema CICS seguido del nombre de transacción del MCA.
- La razón por la que se direcciona el mensaje a la cola de mensajes no entregados se encuentra en el campo *Reason* de la cabecera de cola de mensajes no entregados.
- La cabecera de cola de mensajes no entregados contiene detalles del nombre de la cola de destino y el nombre del gestor de colas.
- La cabecera de cola de mensajes no entregados contiene campos que deben restablecerse en el descriptor de mensaje antes de que el mensaje se coloque en la cola de destino. Son las siguientes:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- El descriptor de mensaje es el mismo que ha puesto (PUT) la aplicación original, excepto los tres campos mostrados (Codificación, CodedCharSetId y Format).

La aplicación de cola de mensajes no entregados debe realizar una o varias de las siguientes acciones:

- Examine el campo *Reason*. Un MCA puede haber puesto un mensaje por las razones siguientes:
  - El mensaje era más largo que el tamaño de mensaje máximo del canal

    La razón es MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - El mensaje no se ha podido poner en la cola de destino

    La razón es cualquier código de razón MQRC_* que puede devolver una operación MQPUT
  - Una salida de usuario ha solicitado esta acción

    El código de razón es el que proporciona la salida de usuario o el valor predeterminado MQRC_SUPPRESSED_BY_EXIT
- Intente reenviar el mensaje a su destino previsto, cuando sea posible.
- Retenga el mensaje durante un determinado periodo de tiempo antes de descartarlo cuando se determina la razón del desvío, pero no se corrige inmediatamente.
- Proporcione instrucciones a los administradores para corregir los problemas, si se han determinado.
- Descarte los mensajes que estén dañados o que no puedan procesarse de otro modo.

Hay dos maneras de manejar los mensajes que se han recuperado de la cola de mensajes no entregados:

1. Si el mensaje es para una cola local:
   - Realice las conversiones de código necesarias para extraer los datos de aplicación
   - Realice conversiones de código en los datos si es una función local
   - Coloque el mensaje resultante en la cola local con todos los detalles del descriptor de mensajes restaurado
2. Si el mensaje es para una cola remota, coloque el mensaje en la cola.

Para obtener información sobre cómo se manejan los mensajes no entregados en un entorno de gestión de colas distribuidas, consulte ¿Qué ocurre cuando no se puede entregar un mensaje?.

Programación de Multicast

Utilice esta información para informarse sobre las tareas de programación de IBM MQ Multicast tales como conectar con un gestor de colas y la generación de informes de excepciones.

IBM MQ Multicast se ha diseñado para que tan transparente como sea posible para el usuario y para que siga siendo compatible con las aplicaciones existentes. La definición de un objeto COMMINFO y la configuración de los parámetros **MCAST** y **COMMINFO** de un objeto TOPIC posibilitan que las aplicaciones IBM MQ no requieran una recodificación significativa para usar la multidifusión. No obstante, puede que haya que tener en cuenta alguna limitación (consulte “Multicast y MQI” en la página 1070 para obtener información adicional) y algunas cuestiones de seguridad (consulte Seguridad de Multicast para obtener información adicional).

Multicast y MQI

Utilice esta información para comprender los conceptos importantes de MQI (Message Queue Interface) y cómo están relacionados con IBM MQ Multicast.

Las suscripciones de Multicast no son duraderas. Dado que no hay colas físicas implicadas, no existe ningún lugar para almacenar los mensajes fuera de línea creados por las suscripciones duraderas.

Una vez se ha suscrito una aplicación a un tema de Multicast, se devuelve a un manejador de objetos que puede consumirlo o ejecutar MQGET desde el mismo como si fuera el manejador de una cola. Esto significa que solo las suscripciones de Multicast gestionadas (suscripciones creadas con MQSO_MANAGED) están soportadas, es decir, no es posible realizar una suscripción y "apuntar" a los mensajes en una cola. Esto significa que los mensajes deben consumirse desde el manejador de objetos devuelto en la llamada de suscripción. En el cliente, los mensajes se almacenan en un almacenamiento intermedio de mensajes hasta que los consume el cliente. Consulte la sección Stanza MessageBuffer del archivo de configuración del cliente para obtener más información. Si el cliente no mantiene la tasa de publicación, se descartan los mensajes, según sea necesario, siendo los mensajes más antiguos los que se descartan en primer lugar.

Normalmente es decisión de la administración si una aplicación utiliza Multicast o no, lo cual se especifica estableciendo el atributo MCAST de un objeto TOPIC. Si una aplicación de publicación debe asegurarse de que no se utiliza Multicast, puede utilizar la opción `MQOO_NO_MULTICAST`. De forma similar, una aplicación suscriptora puede asegurarse de que no se utiliza Multicast mediante una suscripción con la opción `MQSO_NO_MULTICAST`.

IBM MQ Multicast permite utilizar los selectores de mensajes. Una aplicación utiliza un selector para registrar su interés solo en aquellos mensajes con propiedades que satisfacen la consulta SQL92 que representa la serie de selección. Si desea más información sobre selectores de mensajes, consulte “Selectores” en la página 31.

La tabla siguiente contiene una lista de todos los conceptos principales de MQI y cómo se relacionan con Multicast:

Tabla 154. Conceptos de MQI y cómo se relacionan con Multicast

Concepto MQI	**Acción cuando se intenta mediante Multicast**	**Código de razón**
Transferir un mensaje de longitud cero	Se rechaza	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Agrupación	Se rechaza	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentación	Se rechaza	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED

Tabla 154. Conceptos de MQI y cómo se relacionan con Multicast (continuación)

Concepto MQI	Acción cuando se intenta mediante Multicast	Código de razón
Listas de distribución	Se rechaza	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	Se rechaza para manejadores de temas: MQINQ y MQSET de temas no están soportados.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	Aceptado para el manejador gestionado. Solo se puede consultar en `Current Depth`.	• Si el valor es Current Depth, no hay ningún código de razón aplicable. • Si el valor es cualquier otro, salvo Current Depth, el código de razón es 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR.
MQSET	Se rechaza para todos los manejadores.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
Transacciones (XA o no)	Se rechaza	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
Examinar mensaje	Se rechaza	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Bloquear mensajes	Se rechaza	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Examinar con marca	Se rechaza	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Pasar contexto	Se rechaza	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Se rechaza. No es válido intentar y ejecutar MQPUT1 en un solo tema de Multicast.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
Suscripción duradera	Se rechaza si el tema está marcado como "Solo Multicast"; de lo contrario se realiza una suscripción no de Multicast.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED

Tabla 154. Conceptos de MQI y cómo se relacionan con Multicast (continuación)

Concepto MQI	Acción cuando se intenta mediante Multicast	Código de razón
TopicString > 255	Se rechaza. Si la serie de tema tiene más de 255 caracteres, se rechaza en el cliente.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
Suscripción no gestionada realizada	Se rechaza si el tema está marcado como "Solo Multicast"; de lo contrario se realiza una suscripción no de Multicast.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Se rechaza	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Los elementos siguientes expanden algunos de los conceptos de MQI de la tabla anterior, y proporcionan información sobre algunos de los conceptos de MQI que no están en la tabla:

Persistencia de los mensajes
Para los suscriptores de Multicast no duradera, los mensajes persistentes del publicador se entregan de forma irrecuperable.

Recorte de mensaje
El recorte de mensaje está soportado, lo que significa que es posible que una aplicación:

1. Emita MQGET.
2. Obtenga MQRC_TRUNCATED_MSG_FAILED.
3. Asigne un almacenamiento intermedio mayor.
4. Vuelva a emitir MQGET para recuperar el mensaje.

Caducidad de suscripción
No se da soporte a la caducidad de la suscripción. Se omite cualquier intento de establecer una caducidad.

Alta disponibilidad para multidifusión

Utilice esta información para comprender la operación de igual a igual continua de multidifusión IBM MQ; aunque IBM MQ se conecta a un gestor de colas IBM MQ, los mensajes no fluyen a través de dicho gestor de colas.

Aunque hay que establecer una conexión con un gestor de colas para hacer un MQOPEN o MQSUB del objeto de tema de multidifusión, los propios mensajes no fluyen a través del gestor de colas. Por lo tanto, una vez completados los MQOPEN o MQSUB en el objeto de tema de multidifusión, es posible seguir transmitiendo mensajes de multidifusión incluso si se pierde la conexión con el gestor de colas. Hay dos modos de operación:

Se establece una conexión normal con el gestor de colas
La comunicación por multidifusión es posible mientras exista la conexión con el gestor de colas. Si la conexión falla, se aplican las reglas de MQI normales, por ejemplo, un MQPUT al manejador de objeto de multidifusión devuelve 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Se restablece la conexión cliente con el gestor de colas
La comunicación de multidifusión es posible incluso durante una reconexión. Esto significa que, incluso cuando se interrumpe la conexión con el gestor de colas, la colocación y el consumo de mensajes de multidifusión no se ven afectados. El cliente intenta reconectarse con un gestor de colas y, si dicha reconexión falla, el descriptor de conexión se interrumpe y todas las llamadas MQI, incluyendo las de multidifusión, fallarán. Para obtener más información, consulte: Reconexión automática de cliente

Si alguna aplicación emite explícitamente un MQDISC, se cerrarán todas las suscripciones de multidifusión y los descriptores de objetos.

Operación peer to peer continua de multidifusión

Una de las ventajas de la comunicación peer to peer entre clientes es que los mensajes no tienen que fluir a través del gestor de colas; por lo tanto, si la conexión con el gestor de colas se interrumpe, la transferencia de mensajes continúa. Se aplican las restricciones siguientes a los requisitos de mensajes continuos de este modo:

- La conexión se tiene que realizar con una de las opciones MQCNO_RECONNECT_* para la operación continua. Este proceso significa que, aunque la sesión de comunicaciones se pueda interrumpir, el propio descriptor de conexión no se interrumpe, sino que se encuentra en estado de reconexión. Si la reconexión falla, el descriptor de conexión quedará ahora interrumpido, lo que impedirá cualquier llamada MQI adicional.
- En este modo solo se soportan MQPUT, MQGET, MQINQ y Async Consume. Cualquier verbo MQOPEN, MQCLOSE o MQDISC requiere reconectar con el gestor de colas para completar.
- Los flujos de estado al gestor de colas se paran; por tanto, cualquier estado del gestor de colas podría quedar obsoleto o faltar. Esto significa que los clientes pueden estar enviando y recibiendo mensajes sin haber ningún estado conocido en el gestor de colas. Para obtener más información, consulte: Supervisión de aplicación de multidifusión

Conversión de datos en MQI para mensajería de multidifusión

Utilice esta información para entender cómo funciona la conversión de datos para la mensajería de IBM MQ Multicast.

IBM MQ Multicast es un protocolo sin conexión compartido y, por tanto, no es posible que cada cliente realice solicitudes específicas para la conversión de datos. Todos los clientes suscritos a la misma secuencia de multidifusión reciben los mismos datos binarios; por lo tanto, si es necesaria la conversión de datos de IBM MQ, esta se realiza localmente en cada cliente.

Los datos se convierten en el cliente para el tráfico de multidifusión de IBM MQ. Si se especifica la opción **MQGMO_CONVERT**, la conversión de datos se realiza en la forma solicitada. Los formatos definidos por el usuario necesitan que la salida de conversión de datos esté instalada en el cliente; consulte “Escribir salidas de conversión de datos” en la página 1009 para conocer qué bibliotecas están ahora en los paquetes de cliente y servidor.

Para obtener información sobre la administración de la conversión de datos, consulte Habilitación de la conversión de datos para la mensajería de multidifusión.

Para obtener más información sobre la conversión de datos, consulte Conversión de datos.

Si desea más información sobre las salidas de la conversión de datos y `ClientExitPath`, consulte Stanza ClientExitPath del archivo de configuración de cliente.

Notificación de excepciones de multidifusión

Utilice esta información para obtener información sobre los manejadores de evento de IBM MQ Multicast y la creación de informes de excepciones de IBM MQ Multicast.

IBM MQ Multicast ayuda con la determinación de problemas llamando al manejador de sucesos para que informe de sucesos de multidifusión que se notifican utilizando el mecanismo estándar del manejador de sucesos de IBM MQ.

Un suceso de multidifusión individual puede provocar que se llame más de un suceso IBM MQ porque podría haber varios descriptores de conexión MQHCONN utilizando el mismo transmisor o receptor de multidifusión. No obstante, cada excepción de multidifusión solo provoca la invocación de un único manejador de sucesos por conexión de IBM MQ.

La constante de IBM MQ MQCBDO_EVENT_CALL permite a las aplicaciones registrar una devolución de llamada para solo recibir sucesos IBM MQ y MQCBDO_MC_EVENT_CALL permite registrar una devolución de llamada para recibir solo sucesos de multidifusión. Si se utilizan ambas constantes, se recibirán ambos tipos de suceso.

Solicitud de sucesos de multidifusión

Los sucesos de multidifusión de IBM MQ Multicast usan la constante MQCBDO_MC_EVENT_CALL en el campo cbd.Options. En el ejemplo siguiente se muestra cómo solicitar sucesos de multidifusión:

```
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);
```

Cuando se especifica la opción MQCBDO_MC_EVENT_CALL en el campo cbd.Options, al manejador de sucesos solo se le envían sucesos de IBM MQ Multicast en lugar de sucesos de nivel de conexión. Para solicitar que se envíen ambos tipos de sucesos al manejador de sucesos, la aplicación tiene que especificar la constante MQCBDO_EVENT_CALL en el campo cbd.Options, así como la constante MQCBDO_MC_EVENT_CALL, tal como se muestra en el ejemplo siguiente:

```
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);
```

Si no se utiliza ninguna de estas constantes, solo se envían sucesos de nivel de conexión al manejador de sucesos.

Para obtener más información sobre los valores del campo Options , consulte Opciones (MQLONG).

Formato de un suceso de multidifusión

Las excepciones de IBM MQ Multicast incluyen información de soporte que se devuelve en el parámetro **Buffer** de la función de devolución de llamada. El puntero **Buffer** apunta a un vector de punteros y el campo MQCBC.DataLength especifica el tamaño, en bytes, del vector. El primer elemento del vector siempre apunta a una breve descripción textual del suceso. Se podrían suministrar más parámetros en función del tipo de suceso. La tabla siguiente lista las excepciones:

Tabla 155. Descripciones de código de suceso de multidifusión

Código de suceso	Descripción	Datos adicionales
MQMCEV_PACKET_LOSS	Pérdida de paquetes irrecuperable	Número de paquetes perdidos
MQMCEV_HEARTBEAT_TIMEOUT	Larga ausencia del paquete de control de pulsaciones (heartbeat)	No disponible
MQMCEV_VERSION_CONFLICT	Recepción de paquetes de versiones de protocolo más recientes	No disponible
MQMCEV_RELIABILITY	Distintos modos de fiabilidad del transmisor y del receptor	No disponible

Tabla 155. Descripciones de código de suceso de multidifusión (continuación)

Código de suceso	Descripción	Datos adicionales
MQMCEV_CLOSED_TRANS	1 origen ha cerrado la transmisión de tema	No disponible
MQMCEV_STREAM_ERROR	Error detectado en la corriente	No disponible
MQMCEV_NEW_SOURCE	Un origen nuevo empieza a transmitir en el tema	Estructura del origen
MQMCEV_RECEIVE_QUEUE_TRIMMED	Paquetes eliminados de PacketQ por caducidad de tiempo o espacio	Número de paquetes recortados
MQMCEV_PACKET_LOSS_NACK_EXPIRE	Pérdida de paquetes irrecuperable por caducidad de NACK	Número de paquetes perdidos
MQMCEV_ACK_RETRIES_EXCEEDED	Paquetes eliminados del historial tras haberse superado **max_ack_retries** (máximo de reintentos ACK)	Número de paquetes eliminados
MQMCEV_STREAM_SUSPEND_NACK	Los NACK se han suspendido en una corriente aceptada por este tema	ID de corriente de suspensión Tiempo en milisegundos durante el que la corriente se ha suspendido
MQMCEV_STREAM_RESUME_NACK	Los NACK se han reanudado después de haberse suspendido en una corriente	ID de corriente
MQMCEV_STREAM_EXPELLED	Una corriente aceptada por este tema ha sido rechazada debido a una petición de expulsión	ID de corriente
MQMCEV_FIRST_MESSAGE	Primer mensaje de un origen	Número de mensaje
MQMCEV_LATE_JOIN_FAILURE	No se pudo iniciar la sesión de unión tardía	No disponible
MQMCEV_MESSAGE_LOSS	Pérdida de mensajes irrecuperable	Número de mensajes perdidos
MQMCEV_SEND_PACKET_FAILURE	El transmisor de multidifusión no ha podido enviar un paquete de multidifusión	No disponible
MQMCEV_REPAIR_DELAY	El receptor de multidifusión no ha recibido un paquete de reparación de un NAK pendiente	No disponible
MQMCEV_MEMORY_ALERT_ON	Los búfers de recepción del receptor se están llenando	Porcentaje de utilización de búfers
MQMCEV_MEMORY_ALERT_OFF	Los búfers de recepción del receptor han bajado a su nivel normal	Porcentaje de utilización de búfers

Tabla 155. Descripciones de código de suceso de multidifusión (continuación)

Código de suceso	Descripción	Datos adicionales
MQMCEV_NACK_ALERT_ON	La velocidad de petición de paquete de reparación de receptor ha alcanzado la marca de límite superior	Velocidad de petición de reparación actual, en paquetes por segundo
MQMCEV_NACK_ALERT_OFF	La velocidad de petición de paquete de reparación de receptor ha bajado al nivel normal	Velocidad de petición de reparación actual, en paquetes por segundo
MQMCEV_REPAIR_ALERT_ON	La velocidad de envío de paquete de reparación de transmisor ha alcanzado la marca de límite superior	No disponible
MQMCEV_REPAIR_ALERT_OFF	La velocidad de envío de paquete de reparación de transmisor ha bajado al nivel normal	No disponible
MQMCEV_SHM_DEST_UNUSABLE	Se ha detectado que no se puede usar la región de memoria compartida utilizada por un destino de tema de transmisor.	No disponible
MQMCEV_SHM_PORT_UNUSABLE	Se ha detectado que el puerto de memoria compartida utilizado por una instancia de receptor no se puede utilizar	No disponible
MQMCEV_CCT_GETTIME_FAILED	Ha fallado la acción de obtener la hora de la Hora de clúster coordinado.	No disponible
MQMCEV_DEST_INTERFACE_FAILURE	La interfaz de red utilizada por un destino de tema de transmisor ha fallado y no se dispone de una copia de seguridad de la interfaz de red	
MQMCEV_DEST_INTERFACE_FAILOVER	La interfaz de red utilizada por un destino de tema de transmisor ha fallado y se ha completado satisfactoriamente la migración tras error a otra interfaz	
MQMCEV_PORT_INTERFACE-FAILURE	La interfaz de red utilizada por un rmmPort receptor ha fallado y no se dispone de una copia de seguridad de la interfaz de red (o también ha fallado)	Configuración de RMM
MQMCEV_PORT_INTERFACE_FAILOVER	La interfaz de red utilizada por un rmmPort receptor ha fallado y se ha completado satisfactoriamente la migración tras error a otra interfaz	Configuración de RMM

Codificación en C

Tenga en cuenta la información de las secciones siguientes al escribir programas IBM MQ en C.

- "Parámetros de las llamadas MQI" en la página 1077
- "Parámetros con tipo de datos no definido" en la página 1077
- "Tipos de datos" en la página 1077
- "Manipulación de series binarias" en la página 1077
- "Manipulación de series de caracteres" en la página 1078
- "Valores iniciales para estructuras" en la página 1078
- "Valores iniciales para estructuras dinámicas" en la página 1079
- "Utilizar desde C++" en la página 1079

Parámetros de las llamadas MQI

Los parámetros que son *solo de entrada* y de tipo MQHCONN, MQHOBJ, MQHMSG o MQLONG se pasan por valor; para todos los demás parámetros, la *dirección* del parámetro se pasa por valor.

No todos los parámetros que se pasan por dirección deben especificarse cada vez que se invoque la función. Donde no se necesite un parámetro concreto, puede especificarse un puntero nulo como parámetro en la invocación de la función, en lugar de la dirección de los datos del parámetro. Los parámetros para los cuales esto es posible se identifican en las descripciones de llamada.

No se devuelve ningún parámetro como valor de la función; en la terminología de C, esto implica que todas las funciones devuelven void.

Los atributos de la función se definen mediante la variable de macro MQENTRY; el valor de esta variable de macro depende del entorno.

Parámetros con tipo de datos no definido

Las funciones MQGET, MQPUT y MQPUT1 tienen cada una un parámetro **Buffer** que tiene un tipo de datos no definido. Este parámetro se utiliza para enviar y recibir los datos de mensaje de la aplicación.

Los parámetros de este tipo se muestran en los ejemplos de C como matrices de MQBYTE. Puede declarar los parámetros de esta forma, pero normalmente es más conveniente declararlos como la estructura que describe el diseño de los datos del mensaje. El parámetro de función se declara como un puntero a void y, por lo tanto, se puede especificar la dirección de cualquier dato como parámetro en la invocación de la función.

Tipos de datos

Todos los tipos de datos se definen con la sentencia `typedef`.

Para cada tipo de datos, se define también el tipo de datos de puntero correspondiente. El nombre del tipo de datos de puntero es el nombre del tipo de datos elemental o de estructura prefijado con la letra P para denotar un puntero. Los atributos del puntero se definen mediante la variable de macro MQPOINTER; el valor de esta variable de macro depende del entorno. El código siguiente ilustra cómo declarar los tipos de datos de puntero:

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

Manipulación de series binarias

Las series de datos binarios se declaran como uno de los tipos de datos de MQBYTEn.

Siempre que copie, compare o establezca campos de este tipo, utilice las funciones C `memcpy`, `memcmp` o `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));
```

No utilice las funciones de series strcpy, strcmp, strncpy ni strncmp, ya que estas no funcionan correctamente con datos declarados como MQBYTE24.

Manipulación de series de caracteres

Cuando el gestor de colas devuelve los datos de caracteres a la aplicación, el gestor de colas siempre rellena los datos de caracteres con espacios en blanco hasta la longitud definida del campo. El gestor de colas no devuelve series terminadas en nulos, pero puede utilizarlas en su entrada. Por lo tanto, al copiar, comparar o concatenar dichas series, utilice las funciones de serie strncpy, strncmp o strncat.

No utilice las funciones de serie que requieren que la serie termine por un nulo (strcpy, strcmp y strcat). Además, no utilice la función strlen para determinar la longitud de la serie; utilice en su lugar la función sizeof para determinar la longitud del campo.

Valores iniciales para estructuras

El archivo de inclusión <cmqc.h> define diversas variables de macro que puede utilizar para proporcionar valores iniciales para las estructuras al declarar instancias de dichas estructuras. Estas variables de macro tienen nombre con el formato MQxxx_DEFAULT, donde MQxxx representa el nombre de la estructura. Utilícelas de este modo:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

Para algunos campos de caracteres, el MQI define valores concretos que son válidos (por ejemplo, para los campos *StrucId* o para el campo *Format* en MQMD). Para cada uno de los valores válidos, se proporcionan dos variables de macro:

- Una variable de macro define el valor como una serie con una longitud, excluyendo el nulo implícito, que coincide exactamente con la longitud definida del campo. En los ejemplos siguientes, el símbolo ¬ representa un único carácter en blanco:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Utilice este formato con las funciones memcpy y memcmp.
- La otra variable de macro define el valor como una matriz de caracteres; el nombre de esta variable de macro es el nombre del formato de serie con el sufijo _ARRAY. Por ejemplo:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Utilice este formato para inicializar el campo cuando se declare una instancia de la estructura con valores distintos a los proporcionados por la variable de macro MQMD_DEFAULT.

Valores iniciales para estructuras dinámicas

Cuando se necesita un número variable de instancias de una estructura, las instancias se crean normalmente en almacenamiento principal obtenido dinámicamente utilizando las funciones calloc o malloc.

Para inicializar los campos en dichas estructuras, se recomienda la técnica siguiente:

1. Declare una instancia de la estructura utilizando la variable de macro MQxxx_DEFAULT adecuada para inicializar la estructura. Esta instancia pasa a ser el *modelo* para otras instancias:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   Codifique las palabras clave static o auto en la declaración para dar a la instancia modelo un tiempo de vida estático o dinámico, según sea necesario.
2. Utilice las funciones calloc o malloc para obtener almacenamiento para una instancia dinámica de la estructura:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. Utilice la función memcpy para copiar la instancia modelo a la instancia dinámica:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                     /* initialize dynamic instance */
   ```

Utilizar desde C++

Para el lenguaje de programación C++, los archivos de cabecera contienen las sentencias adicionales siguientes que se incluyen únicamente cuando se utiliza un compilador de C++:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Codificación en Visual Basic

Información que se debe tener en cuenta al codificar programas de IBM MQ en Microsoft Visual Basic. Visual Basic solo está soportado en Windows.

Nota:

Stabilized A partir de IBM WebSphere MQ 7.0, fuera del entorno de .NET, el soporte de Visual Basic (VB) se ha estabilizado en el nivel de la IBM WebSphere MQ 6.0. La mayoría de las nuevas funciones añadidas a IBM WebSphere MQ 7.0 o posteriores no están disponibles para las aplicaciones VB. Si está programando en VB.NET, utilice las clases de IBM MQ para .NET. Para obtener más información, consulte Desarrollo de aplicaciones .NET.

Deprecated A partir de IBM MQ 9.0, el soporte para Microsoft Visual Basic 6.0 está en desuso. Las clases de IBM MQ para .NET son la tecnología de sustitución recomendada.

Para evitar la conversión no deseada de los datos binarios que pasan entre Visual Basic y IBM MQ, utilice una definición MQBYTE en lugar de MQSTRING. CMQB.BAS define varios nuevos tipos de MQBYTE que

son equivalentes a una definición de byte C y los utiliza en las estructuras de IBM MQ. Por ejemplo, para la estructura MQMD (descriptor de mensaje), MsgId (identificador de mensaje) se define como MQBYTE24.

Visual Basic no tiene un tipo de datos de puntero, por lo que las referencias a otras estructuras de datos de IBM MQ se realizan por desplazamiento, en lugar de mediante puntero. Declare una estructura compuesta formada por las dos estructuras de componentes y especifique la estructura compuesta en la llamada. El soporte de IBM MQ para Visual Basic proporciona una llamada MQCONNXAny para que esto sea posible y que las aplicaciones cliente puedan especificar las propiedades de canal en una conexión de cliente. Acepta una estructura sin tipo (MQCNOCD) en lugar de la estructura MQCNO típica.

La estructura MQCNOCD es una estructura compuesta formada por un MQCNO seguido de un MQCD. Esta estructura se declara en el archivo de cabecera de salidas CMQXB. Utilice la rutina MQCNOCD_DEFAULTS para inicializar una estructura MQCNOCD. Se proporciona un ejemplo que realiza llamadas MQCONNX (amqscnxb.vbp).

MQCONNXAny tiene los mismos parámetros que MQCONNX, excepto que el parámetro **ConnectOpts** se declara como de Cualquier tipo de datos, en lugar de ser del tipo de datos MQCNO. Esto permite a la función aceptar la estructura MQCNO o la estructura MQCNOCD. Esta función se declara en el archivo de cabecera principal CMQB.

Conceptos relacionados

“Preparación de programas Visual Basic en Windows” en la página 1045
Información a tener en cuenta cuando se utilizan programas de Microsoft Visual Basic en Windows.

Referencia relacionada

“Enlace de aplicaciones de Visual Basic con el código de IBM MQ MQI client” en la página 944
Puede enlazar aplicaciones de Microsoft Visual Basic con el código de IBM MQ MQI client en Windows.

Desarrollo en COBOL

Tenga en cuenta la información de la siguiente sección cuando desarrolle programas IBM MQ en COBOL.

Constantes con nombre

Los nombres de las constantes contienen el carácter de subrayado (_) en el nombre. En COBOL, hay que utilizar el carácter de guión (-) en lugar del carácter de subrayado. Las constantes que tienen valores de cadena de caracteres utilizan el carácter de comillas simples (') como delimitador de cadena. Para hacer que el compilador acepte este carácter, utilice la opción de compilador APOST.

El archivo de copia CMQV contiene declaraciones de las constantes con nombre como elementos de nivel 10. Para utilizar las constantes, declare explícitamente el elemento de nivel 01 y luego utilice la sentencia COPY para copiar en las declaraciones de las constantes:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Sin embargo, este método hace que las constantes ocupen almacenamiento en el programa incluso si no se referencian. Si las constantes se incluyen en muchos programas distintos dentro de la misma unidad de ejecución, existirán múltiples copias de las constantes; esto puede dar lugar a que se utilice una cantidad significativa de almacenamiento principal. Esto puede evitarse añadiendo la cláusula GLOBAL a la declaración de nivel 01:

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Esto solo asigna almacenamiento para *un* único conjunto de constantes dentro de la unidad de ejecución; sin embargo, las constantes pueden ser referenciadas por *cualquier* programa dentro de la unidad de ejecución, no solo por el programa que contiene la declaración de nivel 01.

Aseguramiento de la alineación de estructuras

Hay que asegurarse de que las estructuras de IBM MQ que se pasan para iniciar la llamada MQ estén alineadas en los límites de palabra. Un límite de palabra es de 4 bytes para procesos de 32-bits, 8 bytes para procesos de 64 bits y 16 bytes para procesos de 128 bits (IBM i).

En la medida de lo posible, coloque todas las estructuras IBM MQ juntas para que estén alineadas por límite.

Codificación en lenguaje ensamblador System/390 (interfaz de cola de mensajes)

Tenga en cuenta la información de las secciones siguientes al codificar programas IBM MQ for z/OS en el lenguaje ensamblador.

- "Nombres" en la página 1081
- "Uso de llamadas MQI" en la página 1081
- "Declaración de constantes" en la página 1081
- "Especificación del nombre de una estructura" en la página 1082
- "Especificación del formato de una estructura" en la página 1082
- "Control del listado" en la página 1083
- "Especificación del valor inicial de un campo" en la página 1083
- "Desarrollo de programas reentrantes" en la página 1083
- "Utilización de CEDF" en la página 1084

Nombres

Los nombres de los parámetros en las descripciones de las llamadas y los nombres de los campos de las descripciones de las estructuras aparecen en mayúsculas y minúsculas. En las macros del lenguaje ensamblador proporcionadas con IBM MQ, todos los nombres están en mayúsculas.

Uso de llamadas MQI

La MQI es una interfaz de llamada, por lo que los programas de lenguaje ensamblador deben cumplir el convenio de enlace de SO.

Concretamente, antes de emitir una llamada MQI, los programas de lenguaje ensamblador deben apuntar el registro R13 a un área de guardado de al menos 18 palabras completas. Esta área de guardado proporciona almacenamiento para el programa llamado. Almacena los registros del invocador antes de que se destruya su contenido y restaura el contenido de los registros del invocador al retornar.

Nota: Esto es importante para los programas del lenguaje ensamblador de CICS que utilizan la macro DFHEIENT para configurar su almacenamiento dinámico, pero que optan por alterar temporalmente el DATAREG predeterminado de R13 a otros registros. Cuando la interfaz del gestor de recursos CICS recibe el control desde el apéndice, guarda el contenido actual de los registros en la dirección a la que apunta R13. Si no se reserva un área de almacenamiento para esta finalidad se generan resultados imprevisibles y, probablemente, se provocará una terminación anómala en CICS.

Declaración de constantes

La mayoría de las constantes se declaran como igualdades en la macro CMQA.

Sin embargo, las constantes siguientes no se pueden definir como igualdades y no se incluyen cuando se llama a la macro utilizando las opciones predeterminadas:

- MQACT_NONE
- MQCI_NONE

- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

Para incluirlas, añada la palabra clave EQUONLY=NO cuando invoque la macro.

CMQA está protegido frente a declaraciones múltiples, por lo que se puede incluir muchas veces. Sin embargo, la palabra clave EQUONLY solo entra en vigor la primera vez que se incluye la macro.

Especificación del nombre de una estructura

Para permitir la declaración de más de una instancia de una estructura, la macro que genera la estructura prefija el nombre de cada campo con una cadena de caracteres especificables por el usuario y un carácter de subrayado (_).

Especifique la cadena cuando invoque la macro. Si no especifica una cadena, la macro utilizará el nombre de la estructura para construir el prefijo:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

Las declaraciones de estructura en Descripciones de llamadas muestran el prefijo predeterminado.

Especificación del formato de una estructura

Las macros pueden generar declaraciones de estructura de dos formas, controladas por el parámetro DSECT:

DSECT=YES
Se utiliza una instrucción DSECT de lenguaje ensamblador para iniciar una sección de datos nueva; la definición de estructura sigue inmediatamente a la sentencia DSECT. No se ha asignado ningún almacenamiento, por lo que no es posible ninguna inicialización. La etiqueta en la invocación de macro se usa como nombre de la sección de datos; si no se especifica ninguna etiqueta, se usará el nombre de la estructura.

DSECT=NO
Las instrucciones DC del lenguaje ensamblador se utilizan para definir la estructura en la posición actual de la rutina. Los campos se inicializan con valores, que se pueden especificar codificando los parámetros relevantes en la invocación de la macro. Los campos para los que no se especifican valores en la invocación de la macro se inicializan con valores predterminados.

Se presupone DSECT=NO cuando no se especifica el parámetro DSECT.

Control del listado

Se puede controlar el aspecto de la declaración de estructura en el listado del lenguaje ensamblador con el parámetro LIST:

LIST=YES
La declaración de estructura aparece en el listado de lenguaje ensamblador.

LIST=NO
La declaración de estructura no aparece en el listado de lenguaje ensamblador. Esto es lo predeterminado cuando no se especifica el parámetro LIST.

Especificación del valor inicial de un campo

Se puede especificar el valor que se va a usar para inicializar un campo de una estructura codificando el nombre de dicho campo (sin el prefijo) como un parámetro en la invocación de la macro, acompañado del valor necesario.

Por ejemplo, para declarar una estructura de descriptor de mensaje con el campo *MsgType* inicializado con MQMT_REQUEST, y el campo *ReplyToQ* inicializado con la cadena MY_REPLY_TO_QUEUE, utilice el código siguiente:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

Si especifica una constante nombrada (o igualdad) como un valor en la invocación de la macro, utilice la macro CMQA para definir la constante nombrada. Los valores que sean cadenas de caracteres no deberán encerrarse entre comillas simples (' ').

Desarrollo de programas reentrantes

IBM MQ utiliza sus estructuras para tanto la entrada como la salida. Si desea que su programa siga siendo reentrante:

1. Defina las versiones de almacenamiento de trabajo como DSECT o defina las estructuras en línea dentro de una DSECT ya definida. A continuación, copie la DSECT en el almacenamiento que se obtiene utilizando:
   - Para los programas TSO y los lotes, las macros de ensamblador STORAGE o GETMAIN de z/OS
   - Para CICS, el almacenamiento de trabajo DSECT (DFHEISTG) o el mandato EXEC CICS GETMAIN

   Para inicializar correctamente estas estructuras de almacenamiento de trabajo, copie una versión de constante de la correspondiente estructura en la versión de almacenamiento de trabajo.

   Nota: Las estructuras MQMD y MQXQH tienen una longitud de más de 256 bytes cada una. Para copiar estas estructuras en el almacenamiento, utilice la instrucción de ensamblador MVCL.
2. Reserve espacio de almacenamiento usando el formato LIST (MF=L) de la macro CALL. Cuando utilice la macro CALL para hacer una llamada MQI, utilice el formato EXECUTE (MF=E) de la macro utilizando el almacenamiento reservado anteriormente, tal como se muestra en el ejemplo de “Utilización de CEDF” en la página 1084. Para obtener más ejemplos sobre cómo hacer esto, consulte los programas de ejemplo de lenguaje ensamblador, tal como se suministran con IBM MQ.

Utilice la opción RENT del lenguaje ensamblador para ayudarle a determinar si su programa es reentrante.

Para obtener información sobre cómo escribir programas reenterables, consulte *z/OS MVS Application Development Guide: Assembler Language Programs*.

Utilización de CEDF

Si desea utilizar la transacción proporcionada por CICS, CEDF (CICS Execution Diagnostic Facility) para ayudarle a depurar el programa, añada la palabra clave ,VL a cada sentencia CALL, por ejemplo:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

El ejemplo anterior es un código en lenguaje ensamblador reentrante en el que PARMAREA es un área del almacenamiento de trabajo especificado.

Uso de llamadas MQI

La MQI es una interfaz de llamada, por lo que los programas de lenguaje ensamblador deben cumplir el convenio de enlace de SO. Concretamente, antes de emitir una llamada MQI, los programas de lenguaje ensamblador deben apuntar el registro R13 a un área de guardado de al menos 18 palabras completas. Esta área de guardado proporciona almacenamiento para el programa llamado. Almacena los registros del invocador antes de que se destruya su contenido y restaura el contenido de los registros del invocador al retornar.

Nota: Esto es importante para los programas del lenguaje ensamblador de CICS que utilizan la macro DFHEIENT para configurar su almacenamiento dinámico, pero que optan por alterar temporalmente el DATAREG predeterminado de R13 a otros registros. Cuando la interfaz del gestor de recursos CICS recibe el control desde el apéndice, guarda el contenido actual de los registros en la dirección a la que apunta R13. Si no se reserva un área de almacenamiento adecuada para este fin, se obtendrán resultados imprevisibles y probablemente provocará una terminación anómala en CICS.

IBM i Desarrollo de programas IBM MQ en RPG (solo IBM i)

En la documentación de IBM MQ, los parámetros de las llamadas, los nombres de los tipos de datos, los campos de estructuras y los nombres de constantes se describen todos utilizando sus nombres largos. En RPG, estos nombres se acortan a un máximo de seis caracteres en mayúscula.

Por ejemplo, el campo *MsgType* pasa a ser *MDMT* en RPG. Para obtener más información, consulte la publicación IBM i Application Programming Reference (ILE/RPG).

Codificación en PL/I (solo z/OS)

Información útil cuando se codifica para IBM MQ en PL/I.

Estructuras

Las estructuras se declaran con el atributo BASED y, por tanto, no ocupan almacenamiento a menos que el programa declare una o más instancias de una estructura.

Se puede declarar una instancia de una estructura utilizando el atributo `like`, por ejemplo:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

Los campos de estructura se declaran con el atributo INITIAL; cuando se utiliza el atributo `like` para declarar una instancia de estructura, dicha instancia hereda los valores iniciales definidos para dicha estructura. Solo hay que establecer aquellos campos en los que el valor requerido sea distinto del valor inicial.

En PL/I no se distingue entre mayúsculas y minúsculas, por lo que los nombres de las llamadas, los campos de estructura y las constantes se pueden codificar en minúsculas, mayúsculas o una mezcla de ambas.

Constantes con nombre

Las constantes con nombre se declaran como variables de macro; por tanto, las constantes con nombre no referenciadas en el programa no ocupan ningún almacenamiento en el procedimiento compilado.

Sin embargo, hay que especificar la opción de compilador que hace que el preprocesador de macros procese el código fuente cuando se compila el programa.

Todas las variables de macro son variables de carácter, incluso las que representan valores numéricos. Aunque esto pueda parecer contrario a la intuición, no da lugar a ningún conflicto de tipo de datos después de que las variables de macro hayan sido sustituidas por el procesador de macros, por ejemplo:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Utilización de programas procedimentales de ejemplo de IBM MQ

Estos programas de ejemplo se escriben en lenguajes de procedimiento y muestran los usos típicos de la interfaz de cola de mensajes (MQI). Programas IBM MQ en distintas plataformas.

Acerca de esta tarea

Hay dos conjuntos de ejemplos:

- Programas de ejemplo para sistemas distribuidos e IBM i.
- Programas de ejemplo para z/OS.

Procedimiento

- Utilice los enlaces siguientes para obtener más información sobre los programas de ejemplo:
  - “Utilización de los programas de ejemplo en Multiplataformas” en la página 1086
  - z/OS “Utilización de los programas de ejemplo para z/OS” en la página 1189

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 7
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM MQ. Antes de empezar a diseñar y escribir las aplicaciones IBM MQ, familiarícese con los conceptos básicos de IBM MQ.

“Desarrollo de aplicaciones para IBM MQ” en la página 5
Puede desarrollar aplicaciones para enviar y recibir mensajes, así como para gestionar los gestores de colas y recursos relacionados. IBM MQ admite las aplicaciones escritas en numerosos lenguajes e infraestructuras.

“Consideraciones acerca del diseño de las aplicaciones IBM MQ” en la página 51
Cuando haya decidido cómo pueden beneficiarse sus aplicaciones de las plataformas y entornos que tiene a su disposición, debe decidir cómo utilizar las características que ofrece IBM MQ.

“Desarrollo de una aplicación procedimental para encolamientos” en la página 740
Utilice esta información para obtener información sobre cómo desarrollar aplicaciones de encolamientos, cómo conectarse con un gestor de colas y desconectarse del mismo, sobre publicación/suscripción y apertura y cierre de objetos.

“Desarrollo de aplicaciones procedimentales cliente” en la página 936
Lo que se debe saber para escribir aplicaciones cliente en IBM MQ utilizando un lenguaje procedimental.

“Escritura de aplicaciones de publicación/suscripción” en la página 829
Empezar a escribir aplicaciones de publicación/suscripción de IBM MQ.

“Creación de una aplicación procedimental” en la página 1025
Puede escribir una aplicación IBM MQ en uno de varios lenguajes procedimentales y ejecutar la aplicación en varias plataformas diferentes.

“Tratamiento de errores en un programa procedimental” en la página 1064
Esta información explica los errores asociados a las llamadas MQI de una aplicación cuando se efectúa una llamada o cuando el mensaje se entrega en su destino final.

Multi Utilización de los programas de ejemplo en Multiplataformas

Estos programas de procedimiento de ejemplo se entregan con el producto. Los ejemplos se escriben en C y COBOL y muestran los usos típicos de la interfaz de colas de mensajes (MQI).

Acerca de esta tarea

Los ejemplos no están pensados para mostrar las técnicas de programación generales, por lo que se omite la comprobación de errores que es posible que incluya en un programa de producción.

El código fuente de todos los ejemplos se facilita con el producto; dicho fuente incluye comentarios que explican las técnicas de gestión de colas de mensajes mostradas en los programas.

IBM i Para la programación de RPG, consulte IBM i Application Programming Reference (ILE/RPG).

Los nombres de los ejemplos empiezan por el prefijo `amq`. El cuarto carácter indica el lenguaje de programación y el compilador cuando sea necesario:

- s: lenguaje C
- 0: lenguaje COBOL en compiladores IBM y Micro Focus
- i: lenguaje COBOL solo en compiladores IBM
- m: lenguaje COBOL solo en compiladores Micro Focus

El octavo carácter del ejecutable indica si el ejemplo se ejecuta en la modalidad de enlace local o en la modalidad de cliente. Si no hay un octavo carácter, el ejemplo se ejecuta en modalidad de enlaces locales. Si el octavo carácter es 'c', el ejemplo se ejecuta en modalidad de cliente.

Para poder ejecutar las aplicaciones de ejemplo, antes debe crear y configurar un gestor de colas. Para configurar el gestor de colas para que acepte las conexiones de cliente, consulte “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097.

Procedimiento

- Utilice los enlaces siguientes para obtener más información sobre los programas de ejemplo:
  - “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087
  - “Preparación y ejecución de los programas de ejemplo” en la página 1096
  - “El programa de ejemplo de salida de API” en la página 1103
  - “Programa de ejemplo de consumo asíncrono” en la página 1104
  - “El programa de ejemplo Asynchronous Put (operación de transferencia asíncrona)” en la página 1105
  - “Los programas de ejemplo de examen” en la página 1106
  - “El programa de ejemplo del navegador” en la página 1107
  - “Ejemplo de transacción CICS” en la página 1108
  - “El programa de ejemplo de conexión” en la página 1108
  - “El programa de ejemplo de conexión de datos” en la página 1110
  - “Ejemplos de coordinación de bases de datos” en la página 1110
  - “Ejemplo de cola de mensajes no entregados” en la página 1117

- “El programa de ejemplo de lista de distribución” en la página 1117
- “Los programas de ejemplo de eco” en la página 1118
- “Programas de ejemplo de obtención” en la página 1119
- “Programas de ejemplo de alta disponibilidad” en la página 1121
- “Programas de ejemplo de consulta” en la página 1125
- “Programa de ejemplo de consulta de propiedades de un manejador de mensajes” en la página 1126
- “Los programas de ejemplo de publicación/suscripción” en la página 1126
- “El programa de ejemplo Publish exit (salida de publicación)” en la página 1131
- “Programas de transferencia de ejemplo” en la página 1132
- “Programas de ejemplo de mensaje de referencia” en la página 1134
- “Programas de ejemplo de solicitud” en la página 1141
- “Programas Set de ejemplo” en la página 1147
- “El programa de ejemplo TLS” en la página 1148
- “Programas de ejemplo de desencadenamiento” en la página 1151
- “Utilización de los ejemplos de TUXEDO en AIX, Linux, and Windows” en la página 1153
- “Utilización de la salida de seguridad SSPI en Windows” en la página 1164
- “Ejecución de los ejemplos utilizando colas remotas” en la página 1165
- “El programa de ejemplo Cluster Queue Monitoring (AMQSCLM)” en la página 1165
- “Programa de ejemplo para Connection Endpoint Lookup (CEPL)” en la página 1174

Conceptos relacionados

“Programas de ejemplo C++” en la página 546
Se proporcionan cuatro programas de ejemplo que muestran cómo obtener y transferir mensajes.

Tareas relacionadas

“Utilización de los programas de ejemplo para z/OS” en la página 1189
Las aplicaciones de procedimientos de ejemplo que se entregan con IBM MQ for z/OS ilustran los usos típicos de la interfaz de cola de mensajes (MQI).

Multi *Funciones que se ilustran en los programas de ejemplo en Multiplatforms*

Una colección de tablas que muestran las técnicas demostradas por los programas de ejemplo de IBM MQ.

Todos los ejemplos abren y cierran colas con las llamadas MQOPEN y MQCLOSE, por lo que estas técnicas no se listan por separado en las tablas. Consulte la cabecera que incluya la plataforma en la que esté interesado.

z/OS Para la plataforma z/OS, consulte “Utilización de los programas de ejemplo para z/OS” en la página 1189.

Linux AIX *Ejemplos para sistemas AIX and Linux*

Técnicas ilustradas en los programas de ejemplo para IBM MQ for AIX or Linux.

Consulte “Preparación y ejecución de programas de ejemplo en AIX and Linux” en la página 1100 para averiguar dónde se almacenan los programas de ejemplo para IBM MQ for AIX or Linux .

Tabla 156 en la página 1088 La tabla lista qué archivos fuente C y COBOL se proporcionan, y si se incluye un ejecutable de servidor o cliente.

Tabla 156. Programas de ejemplo que muestran el uso de MQI (C y COBOL) en AIX and Linux.

Una tabla con cuatro columnas. Las primera columna enumera las técnicas ilustradas en los ejemplos. La segunda columna enumera los ejemplos de C y la tercera columna enumera los ejemplos de COBOL que muestran cada una de las técnicas enumeradas en la primera columna. La cuarta columna muestra si un ejecutable C de servidor está o no incluido y la quinta columna muestra si un ejecutable C de cliente está o no incluido.

Técnica	**C (fuente) ("1" en la página 1090)**	**COBOL (fuente) ("2" en la página 1090)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
Utilización de la interfaz de publicación/ suscripción	amqspuba amqssuba amqssbxa	no hay ejemplo	amqspub amqssub amqssbx	no hay ejemplo
Colocación de mensajes mediante la llamada MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Colocación de un único mensaje utilizando la llamada MQPUT1	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Colocación de mensajes en una lista de distribución ("3" en la página 1090)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Respuesta a un mensaje de solicitud	amqsinqa	amqminqx amqiinqx	amqsinq	no hay ejemplo
Obtención de mensajes usando una exploración (sin espera)	amqsgbr0	amq0gbr0	amqsgbr	no hay ejemplo
Obtención de mensajes (espera con límite de tiempo)	amqsget0	amq0get0	amqsget	amqsgetc
Obtención de mensajes (espera ilimitada)	amqstrg0	no hay ejemplo	amqstrg	amqstrgc
Obtención de mensajes (con conversión de datos)	amqsecha	no hay ejemplo	amqsech	no hay ejemplo
Colocación de mensajes de referencia en una cola ("3" en la página 1090)	amqsprma	no hay ejemplo	amqsprm	amqsprmc
Obtención de mensajes de referencias de una cola ("3" en la página 1090)	amqsgrma	no hay ejemplo	amqsgrm	amqsgrmc
Salida de canal de mensaje de referencia ("3" en la página 1090)	amqsqrma amqsxrma	no hay ejemplo	amqsxrm	no hay ejemplo
Exploración de los primeros 20 caracteres de un mensaje	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Exploración de mensajes completos	amqsbcg0	no hay ejemplo	amqsbcg	amqsbcgc
Utilización de una cola de entrada compartida	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
Utilización de una cola de entrada exclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilización de la llamada MQINQ	amqsinqa	amqminqx amqiinqx	amqsinq	no hay ejemplo

Tabla 156. Programas de ejemplo que muestran el uso de MQI (C y COBOL) en AIX and Linux.

Una tabla con cuatro columnas. Las primera columna enumera las técnicas ilustradas en los ejemplos. La segunda columna enumera los ejemplos de C y la tercera columna enumera los ejemplos de COBOL que muestran cada una de las técnicas enumeradas en la primera columna. La cuarta columna muestra si un ejecutable C de servidor está o no incluido y la quinta columna muestra si un ejecutable C de cliente está o no incluido.

(continuación)

Técnica	**C (fuente) ("1" en la página 1090)**	**COBOL (fuente) ("2" en la página 1090)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
Utilización de la llamada MQSET	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
Utilización de una cola de respuesta	amqsreq0	amq0req0	amqsreq	amqsreqc
Solicitud de excepciones de mensaje	amqsreq0	amq0req0	amqsreq	no hay ejemplo
Aceptación de un mensaje truncado	amqsgbr0	amq0gbr0	amqsgbr	no hay ejemplo
Utilización de un nombre de cola resuelto	amqsgbr0	amq0gbr0	amqsgbr	no hay ejemplo
Desencadenamiento de un proceso	amqstrg0	no hay ejemplo	amqstrg	amqstrgc
Utilización de la conversión de datos	("4" en la página 1090)	no hay ejemplo	no hay ejemplo	no hay ejemplo
IBM MQ (coordinando gestores de bases de datos XA) accediendo a una única base de datos mediante SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	no hay ejemplo	no hay ejemplo
IBM MQ (coordinando gestores de bases de datos XA) accediendo a dos bases de datos mediante SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	no hay ejemplo	no hay ejemplo
Transacción CICS ("5" en la página 1090)	amqscic0.ccs	no hay ejemplo	amqscic0	no hay ejemplo
Transacción Encina ("3" en la página 1090)	amqsxae0	no hay ejemplo	amqsxae0	no hay ejemplo
Transacción TUXEDO para colocar mensajes "6" en la página 1091)	amqstxpx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Transacción TUXEDO para obtener mensajes ("6" en la página 1091)	amqstxgx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Servidor para TUXEDO ("6" en la página 1091)	amqstxsx	no hay ejemplo	no hay ejemplo	no hay ejemplo

Tabla 156. Programas de ejemplo que muestran el uso de MQI (C y COBOL) en AIX and Linux.

Una tabla con cuatro columnas. Las primera columna enumera las técnicas ilustradas en los ejemplos. La segunda columna enumera los ejemplos de C y la tercera columna enumera los ejemplos de COBOL que muestran cada una de las técnicas enumeradas en la primera columna. La cuarta columna muestra si un ejecutable C de servidor está o no incluido y la quinta columna muestra si un ejecutable C de cliente está o no incluido.

(continuación)

Técnica	**C (fuente) ("1" en la página 1090)**	**COBOL (fuente) ("2" en la página 1090)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
Manejador de cola de mensajes no entregados	Directorio `./tools/c/Samples/dlq` ("7" en la página 1091)	no hay ejemplo	amqsdlq	no hay ejemplo
Colocación de un mensaje desde un cliente MQI	no hay ejemplo	no hay ejemplo	no hay ejemplo	amqsputc
Obtención de un mensaje desde un cliente MQI	no hay ejemplo	no hay ejemplo	no hay ejemplo	amqsgetc
Conexión con el gestor de colas utilizando MQCONNX	amqscnxc	no hay ejemplo	no hay ejemplo	amqscnxc
Utilización de salidas de API	amqsaxe0	no hay ejemplo	amqsaxe	no hay ejemplo
Salida de equilibrado de cargas de trabajo en un clúster	amqswlm0	no hay ejemplo	amqswlm	no hay ejemplo
Colocación de mensajes de forma asíncrona y obtención del estado con la llamada MQSTAT	amqsapt0	no hay ejemplo	amqsapt	amqsaptc
Clientes que se pueden volver a conectar	amqsphac amqsghac amqsmhac	no hay ejemplo	no aplicable	amqsphac amqsghac amqsmhac
Utilización de consumidores de mensajes para consumir mensajes de varias colas asíncronamente	amqscbf0	no hay ejemplo	amqscbf	amqscbfc
Especificación de la información de conexión TLS en MQCONNX	amqssslc	no hay ejemplo	no aplicable	amqssslc

Notas:

1. Los ejemplos de la versión ejecutable de IBM MQ MQI client comparten el mismo código fuente que los ejemplos que se ejecutan en un entorno de servidor.
2. Compile los programas que empiezan por 'amqm' con el compilador Micro Focus de COBOL, los que empiecen por 'amqi' con el compilador de COBOL de IBM y los que empiezan por 'amq0' con cualquiera de ellos.
3. AIX Solo se admite en IBM MQ for AIX .
4. AIX En IBM MQ for AIX este programa se denomina `amqsvfc0.c`
5. AIX CICS solo está soportado en IBM MQ for AIX.

6. Linux TUXEDO no está soportado por IBM MQ para Linux en System p.
7. El origen del manejador de colas de mensajes no entregados consta de varios archivos y se proporciona en un directorio aparte.

Para obtener información detallada sobre el soporte en sistemas AIX and Linux, consulte Requisitos del sistema para IBM MQ.

Windows *Ejemplos para IBM MQ for Windows*

Técnicas ilustradas en los programas de ejemplo para IBM MQ for Windows.

Tabla 157 en la página 1091 lista qué archivos de origen C y COBOL se proporcionan, y si se incluye un ejecutable de servidor o cliente.

Tabla 157. Programas de ejemplo de IBM MQ for Windows que ilustran el uso de la MQI (C y COBOL)

Técnica	**C (fuente)**	**COBOL (fuente)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
Utilización de la interfaz de publicación/ suscripción	amqspuba amqssuba amqssbxa	no hay ejemplo	amqspub amqssub amqssbx	no hay ejemplo
Colocación de mensajes mediante la llamada MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Colocación de un único mensaje utilizando la llamada MQPUT1	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
Colocación de mensajes en una lista de distribución	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Respuesta a un mensaje de solicitud	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Obtención de mensajes (sin espera)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Obtención de mensajes (espera con límite de tiempo)	amqsget0	amq0get0	amqsget	amqsgetc
Obtención de mensajes (espera ilimitada)	amqstrg0	no hay ejemplo	amqstrg	amqstrgc
Obtención de mensajes (con conversión de datos)	amqsecha	no hay ejemplo	amqsech	amqsechc
Colocación de mensajes de referencia en una cola	amqsprma	no hay ejemplo	amqsprm	amqsprmc
Obtención de mensajes de referencia de una cola	amqsgrma	no hay ejemplo	amqsgrm	amqsgrmc
Salida de canal de mensaje de referencia	amqsqrma amqsxrma	no hay ejemplo	amqsxrm	no hay ejemplo
Exploración de los primeros 20 caracteres de un mensaje	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Exploración de mensajes completos	amqsbcg0	no hay ejemplo	amqsbcg	amqsbcgc

Tabla 157. Programas de ejemplo de IBM MQ for Windows que ilustran el uso de la MQI (C y COBOL) (continuación)

Técnica	C (fuente)	COBOL (fuente)	Servidor (ejecutable C)	Cliente (ejecutable C)
Utilización de una cola de entrada compartida	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Utilización de una cola de entrada exclusiva	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilización de la llamada MQINQ	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Utilización de la llamada MQSET	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
Utilización de la llamada MQINQMP	amqsiqma	no hay ejemplo	no hay ejemplo	no hay ejemplo
Utilización de una cola de respuesta	amqsreq0	amq0req0	amqsreq	amqsreqc
Solicitud de excepciones de mensaje	amqsreq0	amq0req0	amqsreq	amqsreqc
Aceptación de un mensaje truncado	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Utilización de un nombre de cola resuelto	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Desencadenamiento de un proceso	amqstrg0	no hay ejemplo	amqstrg	amqstrgc
Utilización de la conversión de datos	amqsvfc0	no hay ejemplo	no hay ejemplo	no hay ejemplo
IBM MQ (coordinando gestores de bases de datos XA) accediendo a una única base de datos mediante SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	no hay ejemplo	no hay ejemplo
IBM MQ (coordinando gestores de bases de datos XA) accediendo a dos bases de datos mediante SQL	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	no hay ejemplo	no hay ejemplo
Transacción TUXEDO para colocar mensajes	amqstxpx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Transacción TUXEDO para obtener mensajes	amqstxgx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Servidor de TUXEDO	amqstxsx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Manejador de cola de mensajes no entregados	Directorio `./tools/c/Samples/dlq` (“1” en la página 1093)	no hay ejemplo	amqsdlq	no hay ejemplo

Tabla 157. Programas de ejemplo de IBM MQ for Windows que ilustran el uso de la MQI (C y COBOL) (continuación)				
Técnica	**C (fuente)**	**COBOL (fuente)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
Colocación de un mensaje desde IBM MQ MQI client	no hay ejemplo	no hay ejemplo	no hay ejemplo	amqsputc
Obtención de un mensaje desde IBM MQ MQI client	no hay ejemplo	no hay ejemplo	no hay ejemplo	amqsgetc
Conexión con el gestor de colas utilizando MQCONNX	amqscnxc	no hay ejemplo	no hay ejemplo	amqscnxc
Utilización de salidas de API	amqsaxe0	no hay ejemplo	amqsaxe	no hay ejemplo
Equilibrado de cargas de trabajo en un clúster	amqswlm0	no hay ejemplo	amqswlm	no hay ejemplo
Rutinas de seguridad SSPI	amqsspin	no hay ejemplo	amqrspin.dll	amqrspin.dll
Colocación de mensajes de forma asíncrona y obtención del estado con la llamada MQSTAT	amqsapt0	no hay ejemplo	amqsapt	amqsaptc
Clientes que se pueden volver a conectar	amqsphac amqsghac amqsmhac	no hay ejemplo	No aplicable	amqsphac amqsghac amqsmhac
Utilización de consumidores de mensajes para consumir mensajes de varias colas asíncronamente	amqscbf0	no hay ejemplo	amqscbf	amqscbfc
Especificación de la información de conexión TLS en MQCONNX	amqssslc	no hay ejemplo	no aplicable	amqssslc

Notas:

1. El origen del manejador de colas de mensajes no entregados consta de varios archivos y se proporciona en un directorio aparte.

Windows *Ejemplos de Visual Basic para IBM MQ for Windows*

Técnicas ilustradas en los programas de ejemplo de IBM MQ en sistemas Windows.

Tabla 158 en la página 1093 muestra las técnicas ilustradas en los programas de ejemplo de IBM MQ for Windows.

Un proyecto puede contener varios archivos. Cuando se abre un proyecto en Visual Basic, los demás archivos se cargan automáticamente. No se proporcionan programas ejecutables.

Todos los proyectos de ejemplo, excepto mqtrivc.vbp, están configurados para funcionar con el servidor IBM MQ. Para descubrir cómo cambiar los proyectos de ejemplo para que funcionen con clientes de IBM MQ, consulte “Preparación de programas Visual Basic en Windows” en la página 1045.

Tabla 158. Programas de ejemplo de IBM MQ for Windows que ilustran el uso de MQI (Visual Basic)	
Técnica	**Nombre de archivo de proyecto**
Colocación de mensajes mediante la llamada MQPUT	amqsputb.vbp

Tabla 158. Programas de ejemplo de IBM MQ for Windows que ilustran el uso de MQI (Visual Basic) (continuación)	
Técnica	**Nombre de archivo de proyecto**
Obtención de mensajes utilizando la llamada MQGET	amqsgetb.vbp
Examen de una cola utilizando la llamada MQGET	amqsbcgb.vbp
Ejemplo sencillo de MQGET y MQPUT (cliente)	mqtrivc.vbp
Ejemplo sencillo de MQGET y MQPUT (servidor)	mqtrivs.vbp
Colocación y obtención de cadenas y estructuras definidas por el usuario usando MQPUT y MQGET	strings.vbp
Utilización de estructuras PCF para iniciar y parar un canal	pcfsamp.vbp
Creación de una cola utilizando la MQAI	amqsaicq.vbp
Listado de las colas de un gestor de colas utilizando la MQAI	amqsailq.vbp
Supervisión de sucesos utilizando la MQAI	amqsaiem.vbp

IBM i *Ejemplos para IBM i*

Técnicas ilustradas en los programas de ejemplo de IBM MQ en sistemas IBM i.

Tabla 159 en la página 1094 muestra las técnicas ilustradas en los programas de ejemplo de IBM MQ for IBM i. Algunas técnicas se utilizan en más de un programa de ejemplo, pero en la tabla se indica un solo programa.

Tabla 159. Programas de ejemplo que demuestran el uso de la MQI (C y COBOL) en IBM i				
Técnica	**C (fuente) ("1" en la página 1096)**	**COBOL (fuente) ("2" en la página 1096)**	**RPG (fuente) ("3" en la página 1096)**	**Cliente (ejecutable C)(4)**
Colocación de mensajes mediante la llamada MQPUT	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
Colocación de mensajes procedentes de un archivo de datos mediante la llamada MQPUT	AMQSPUT4	no hay ejemplo	no hay ejemplo	no hay ejemplo
Colocación de un único mensaje utilizando la llamada MQPUT1	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Colocación de mensajes en una lista de distribución	AMQSPTL4	no hay ejemplo	no hay ejemplo	AMQSPTLC
Respuesta a un mensaje de solicitud	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Obtención de mensajes (sin espera)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Obtención de mensajes (espera con límite de tiempo)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
Obtención de mensajes (espera ilimitada)	AMQSTRG4	no hay ejemplo	AMQ3TRG4	AMQSTRGC
Obtención de mensajes (con conversión de datos)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Colocación de mensajes de referencia en una cola	AMQSPRM4	no hay ejemplo	no hay ejemplo	AMQSPRMC

Tabla 159. Programas de ejemplo que demuestran el uso de la MQI (C y COBOL) en IBM i (continuación)

Técnica	**C (fuente) ("1" en la página 1096)**	**COBOL (fuente) ("2" en la página 1096)**	**RPG (fuente) ("3" en la página 1096)**	**Cliente (ejecutable C)(4)**
Obtención de mensajes de referencia de una cola	AMQSGRM4	no hay ejemplo	no hay ejemplo	AMQSGRMC
Salida de canal de mensaje de referencia	AMQSQRM4, AMQSXRM4	no hay ejemplo	no hay ejemplo	no hay ejemplo
Salida de mensajes	AMQSCMX4	no hay ejemplo	no hay ejemplo	no hay ejemplo
Exploración de los primeros 49 caracteres de un mensaje	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Exploración de mensajes completos	AMQSBCG4	no hay ejemplo	no hay ejemplo	AMQSBCGC
Utilización de una cola de entrada compartida	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilización de una cola de entrada exclusiva	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Utilización de la llamada MQINQ	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilización de la llamada MQSET	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
Utilización de una cola de respuesta	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Solicitud de excepciones de mensaje	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Aceptación de un mensaje truncado	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Utilización de un nombre de cola resuelto	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Desencadenamiento de un proceso	AMQSTRG4	no hay ejemplo	AMQ3TRG4	AMQSTRGC
Servidor de desencadenantes	AMQSERV4	no hay ejemplo	AMQ3SRV4	no hay ejemplo
Utilización de un servidor de desencadenante (incluyendo las transacciones de CICS)	AMQSERV4	no hay ejemplo	AMQ3SRV4	no hay ejemplo
Utilización de la conversión de datos	AMQSVFC4	no hay ejemplo	no hay ejemplo	no hay ejemplo
Utilización de salidas de API	AMQSAXE0	no hay ejemplo	no hay ejemplo	no hay ejemplo
Equilibrado de cargas de trabajo en un clúster	AMQSWLM0	no hay ejemplo	no hay ejemplo	no hay ejemplo
Colocación de mensajes de forma asíncrona y obtención del estado con la llamada MQSTAT	AMQSAPT0	no hay ejemplo	no hay ejemplo	AMQSAPTC
Utilización de la interfaz de publicación/ suscripción	AMQSPUBA, AMQSSUBA, AMQSSBXA	no hay ejemplo	no hay ejemplo	AMQSPUBC, AMQSSUBC, AMQSSBXC
Clientes reconectables (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	no hay ejemplo	no hay ejemplo	no hay ejemplo

Tabla 159. Programas de ejemplo que demuestran el uso de la MQI (C y COBOL) en IBM i (continuación)				
Técnica	**C (fuente) ("1" en la página 1096)**	**COBOL (fuente) ("2" en la página 1096)**	**RPG (fuente) ("3" en la página 1096)**	**Cliente (ejecutable C)(4)**
Utilización de consumidores de mensajes para consumir mensajes de varias colas asíncronamente (5)	AMQSCBFO	no hay ejemplo	no hay ejemplo	no hay ejemplo
Especificación de la información de conexión TLS en MQCONNX	AMQSSSLC	no hay ejemplo	no hay ejemplo	AMQSSSLC
Conexión con el gestor de colas utilizando MQCONNX	AMQSCNXC	no hay ejemplo	no hay ejemplo	AMQSCNXC
Consultar propiedades de un manejador de mensajes, utilizando MQINQMP, desde una cola de mensajes	AMQISQMA	no hay ejemplo	no hay ejemplo	AMQISQMC
Establecer propiedades de un manejador de mensajes utilizando MQSETMP y colocarlo en una cola de mensajes	AMQSSQMA	no hay ejemplo	no hay ejemplo	AMQSSQMC

Notas:

1. Los fuentes de los ejemplos en C se encuentran en el archivo QMQMSAMP/QCSRC. Los archivos de inclusión existen como miembros en el archivo QMQM/H.
2. Los fuentes de los ejemplos en COBOL se encuentran en los archivos QMQMSAMP/QCBLLESRC. Los miembros se llaman AMQ0 xxx 4, donde xxx indica la función de ejemplo.
3. Los fuentes de los ejemplos de RPG están en QMQMSAMP/QRPGLESRC. Los miembros se llaman AMQ3 xxx 4, donde xxx indica la función de ejemplo. Los miembros de copia existen en QMQM/QRPGLESRC. Cada nombre de miembro tiene el sufijo G.
4. Los ejemplos de la versión ejecutable de IBM MQ MQI client comparten el mismo código fuente que los ejemplos que se ejecutan en un entorno de servidor. Los fuentes de los ejemplos en el entorno del cliente son los mismos que los del servidor. Los ejemplos de IBM MQ MQI client están enlazados a la biblioteca de cliente LIBMQIC y los ejemplos del servidor IBM MQ están enlazados a la biblioteca de servidor LIBMQM.
5. Si se tiene que ejecutar el ejecutable del cliente para la aplicación de ejemplo del cliente reconectable y la aplicación de consumidor asíncrono, se tiene que compilar y enlazar a la biblioteca L con hebrasIBMQIC_R. Por tanto, tiene que ejecutar en un entorno con hilos. Establezca la variable de entorno QIBM_MULTI_THREADED en 'Y' y ejecute la aplicación desde qsh.

   Consulte Configuración de IBM MQ con Java y JMS para obtener más información.

Consulte "Preparación y ejecución de programas de ejemplo en IBM i" en la página 1099 para obtener más información.

Además de estas, la opción de ejemplo de IBM MQ for IBM i incluye un archivo de datos de ejemplo, que se utiliza como entrada en los programas de ejemplo, AMQSDATA y los programas JCL de ejemplo que demuestran las tareas de administración. Los ejemplos de CL se describen en Administración de IBM i . Puede utilizar el programa CL de ejemplo amqsamp4 para crear las colas que se usan en los programas de ejemplo descritos en este tema.

Multi

Preparación y ejecución de los programas de ejemplo

Tras completar algunos preparativos iniciales, se pueden ejecutar los programas de ejemplo.

Acerca de esta tarea

Antes de ejecutar los programas de ejemplo, hay que crear un gestor de colas así como las colas necesarias. Puede que también haya que realizar algún preparativo adicional, por ejemplo, si se desea ejecutar los ejemplos de COBOL. Tras completar los preparativos necesarios, se pueden ejecutar los programas de ejemplo.

Procedimiento

Para obtener información sobre cómo preparar y ejecutar los programas de ejemplo, consulte los temas siguientes:

- "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097
- "Preparación y ejecución de programas de ejemplo en IBM i" en la página 1099
- "Preparación y ejecución de programas de ejemplo en AIX and Linux" en la página 1100
- "Preparación y ejecución de programas de ejemplo en Windows" en la página 1101

Multi *Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms*

Para poder ejecutar las aplicaciones de ejemplo, en primer lugar, debe crear un gestor de colas. A continuación, puede configurar el gestor de colas para que acepte de forma segura las solicitudes de conexión de entradas de las aplicaciones que se ejecutan en modo cliente.

Antes de empezar

Asegúrese de que el gestor de colas ya existe y se ha iniciado. Determine si ya se han habilitado los registros de autenticación de canal emitiendo el mandato MQSC:

```
DISPLAY QMGR CHLAUTH
```

Importante: Esta tarea espera a que se habiliten los registros de autenticación de canal. Si se trata de un gestor de colas utilizado por otros usuarios y aplicaciones, cambiar este valor afectará a los demás usuarios y aplicaciones. Si su gestor de colas no utiliza los registros de autenticación de canal, se puede sustituir el paso 4 por un método de autenticación alternativo, tal como una salida de seguridad, que establece MCAUSER en el *non-privileged-user-id* que obtendrá en el paso "1" en la página 1097.

Debe saber el nombre de canal que la aplicación espera utilizar para que se le pueda permitir el uso del canal. También debe saber qué objetos, por ejemplo, colas o temas, espera utilizar la aplicación para que se le pueda permitir utilizarlos.

Acerca de esta tarea

Esta tarea crea un ID de usuario sin privilegios para utilizarlo con una aplicación cliente que se conecta al gestor de colas. El acceso se otorga solo a la aplicación de cliente para que pueda utilizar el canal que necesita y la cola que necesita mediante este ID de usuario.

Procedimiento

1. Obtenga un ID de usuario en el sistema en el que se ejecuta el gestor de colas. En esta tarea, este ID de usuario no debe ser el de un usuario administrativo con privilegios. Este ID de usuario será la autorización bajo la cual se ejecutará la conexión de cliente en el gestor de colas.
2. Inicie un programa de escucha con los mandatos siguientes, donde:

   qmgr-name es el nombre del gestor de colas
   nnnn es el número de puerto elegido

   a) ALW
   Para sistemas AIX, Linux, and Windows :

```
runmqlsr -t tcp -m qmgr-name -p nnnn
```

b) IBM i

Para IBM i:

```
STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
```

3. Si la aplicación utiliza SYSTEM.DEF.SVRCONN>, este canal ya se ha definido. Si la aplicación utiliza otro canal, créelo emitiendo el siguiente mandato MQSC:

```
DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Channel for use by sample programs')
```

donde *nombre-canal* es el nombre del canal.

4. Cree una regla de autenticación de canal que permita que solo la dirección IP del sistema cliente utilice el canal emitiendo el siguiente mandato MQSC:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

donde

channel-name es el nombre del canal.

client-machine-IP-address es la dirección IP del sistema cliente. Si su aplicación de cliente de ejemplo se ejecuta en la misma máquina que el gestor de colas, utilice una dirección IP de '127.0.0.1', si su aplicación se va a conectar utilizando 'localhost'. Si se van a conectar varias máquinas diferentes, puede utilizar un patrón o un rango en lugar de una única dirección IP. Para obtener más información, consulte la sección Direcciones IP genéricas.

non-privileged-user-id es el ID de usuario que ha obtenido en el paso “1” en la página 1097

5. Si la aplicación utiliza SYSTEM.DEFAULT.LOCAL.QUEUE esta cola ya está definida. Si la aplicación utiliza otra cola, créela emitiendo el siguiente mandato MQSC:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

donde *queue-name* es el nombre de su cola.

6. Otorgue acceso para conectarse y consultar el gestor de colas emitiendo el siguiente mandato MQSC:

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

donde *id-usuario-no-privilegiado* es el ID de usuario que ha obtenido en el paso “1” en la página 1097 .

7. Si la aplicación es una aplicación punto a punto, es decir, utiliza colas, otorgue acceso para permitir la consulta y la transferencia y obtención de mensajes utilizando la cola por el ID de usuario que se va a utilizar, emitiendo los siguientes mandatos MQSC:

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

donde

nombre-cola es el nombre de la cola

non-privileged-user-id es el ID de usuario que ha obtenido en el paso “1” en la página 1097

8. Si la aplicación es una aplicación de publicación/suscripción, es decir hace uso de temas, otorgue acceso para permitir la publicación y suscripción utilizando el tema mediante el ID de usuario que se utilizará, emitiendo los mandatos MQSC:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

donde

non-privileged-user-id es el ID de usuario que ha obtenido en el paso “1” en la página 1097
De este modo, el *non-privileged-user-id* tendrá acceso a cualquier tema del árbol de temas, o puede definir un objeto de tema utilizando **DEFINE TOPIC** y otorgar accesos únicamente a la parte del árbol de temas a la que hace referencia dicho objeto de tema. Para obtener más información, consulte la sección Controlar el acceso de usuario a los temas.

Qué hacer a continuación

La aplicación cliente se puede ahora conectar al gestor de colas y transferir u obtener mensajes utilizando la cola.

Conceptos relacionados

ALW Otorgar acceso a un objeto IBM MQ en AIX, Linux, and Windows

Referencia relacionada

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC
IBM i Autorizaciones de IBM MQ en IBM i

IBM i *Preparación y ejecución de programas de ejemplo en IBM i*

Antes de ejecutar los programas de ejemplo en IBM i, en primer lugar, debe crear un gestor de colas y también crear las colas que necesite. Si desea ejecutar ejemplos de COBOL, es posible que necesite algún tipo de preparación adicional.

Acerca de esta tarea

El código fuente de los programas de ejemplo de IBM MQ for IBM i se proporciona en la biblioteca QMQMSAMP como miembros de QCSRC, QCLSRC, QCBLLESRC y QRPGLESRC.

Puede utilizar sus propias colas cuando ejecute los ejemplos o puede ejecutar el programa de ejemplo AMQSAMP4 para crear algunas colas de ejemplo. El código fuente del programa AMQSAMP4 se incluye en el archivo QCLSRC de la biblioteca QMQMSAMP. Puede compilarlo con el comando CRTCLPGM.

Para ejecutar los ejemplos, use las versiones ejecutables en C que se proporcionan en la biblioteca QMQM, o compílelos de forma similar a cualquier aplicación IBM MQ.

Procedimiento

1. Cree un gestor de colas y configure las definiciones predeterminadas.

   Debe hacer esto antes de ejecutar cualquiera de los programas de ejemplo. Para obtener más información sobre cómo crear un gestor de colas, consulte Administración de IBM MQ. Para obtener más información sobre cómo configurar un gestor de colas para aceptar de forma segura las solicitudes de conexión entrantes de las aplicaciones que se ejecutan en la modalidad de cliente, consulte “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097.
2. Para invocar uno de los programas de ejemplo utilizando datos del miembro PUT en el archivo AMQSDATA de la biblioteca QMQMSAMP, utilice un comando como el siguiente:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

Nota: Para que un módulo compilado utilice el sistema de archivos IFS, especifique la opción SYSIFCOPT(*IFSIO) en CRTCMOD y luego el nombre de archivo, pasado como parámetro y con el formato siguiente:

```
home/me/myfile
```

3. Si desea utilizar las versiones COBOL de los ejemplos Inquire (examinar), Set (establecer) y Echo (eco), cambie las definiciones de proceso antes de ejecutar dichos ejemplos.

   Para los ejemplos Inquire, Set y Echo, las definiciones de ejemplo desencadenan las versiones C de estos ejemplos. Si desea las versiones en COBOL, tendrá que cambiar las definiciones de proceso:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   En IBM i, puede utilizar el comando **CHGMQMPRC** (para obtener detalles, consulte Cambiar proceso MQ (CHGMQMPRC)), o edite y ejecute el comando **AMQSAMP4** con la definición alternativa.
4. Ejecute los programas de ejemplo.

   Para obtener más información sobre los parámetros que espera cada uno de los ejemplos, consulte las descripciones de los ejemplos individuales.

   Nota: Para los programas de ejemplo COBOL, cuando pase nombres de cola como parámetros, debe proporcionar 48 caracteres, rellenando con caracteres en blanco si es necesario. Cualquier cantidad distinta a 48 caracteres provocará que el programa falle con código de razón 2085.

Referencia relacionada

"Ejemplos para IBM i" en la página 1094
Técnicas ilustradas en los programas de ejemplo de IBM MQ en sistemas IBM i.

Linux AIX *Preparación y ejecución de programas de ejemplo en AIX and Linux*

Antes de ejecutar los programas de ejemplo en AIX and Linux, en primer lugar, debe crear un gestor de colas y también crear las colas que necesite. Si desea ejecutar ejemplos de COBOL, es posible que necesite algún tipo de preparación adicional.

Acerca de esta tarea

Los archivos de ejemplos de IBM MQ en sistemas AIX and Linux están en los directorios que se indican en Tabla 160 en la página 1100 si se han utilizado los valores predeterminados en el momento de la instalación.

Tabla 160. Dónde encontrar los ejemplos de IBM MQ en sistemas AIX and Linux

Contenido	**Directorio**
archivos fuente	*MQ_INSTALLATION_PATH*/samp
archivos fuente del manejador de cola de mensajes no entregados	*MQ_INSTALLATION_PATH*/samp/dlq
archivos ejecutables	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Los ejemplos necesitan un conjunto de colas con las que trabajar. Puede utilizar sus propias colas o ejecutar el archivo MQSC de ejemplo amqscos0.tst para crear un conjunto. Para ejecutar los ejemplos, utilice las versiones ejecutables proporcionadas o compile las versiones de código fuente como lo haría con cualquier otra aplicación, utilizando un compilador ANSI.

Procedimiento

1. Cree un gestor de colas y configure las definiciones predeterminadas.

   Debe hacer esto antes de ejecutar cualquiera de los programas de ejemplo. Para obtener más información sobre cómo crear un gestor de colas, consulte Administración de IBM MQ. Para obtener más información sobre cómo configurar un gestor de colas para aceptar de forma segura las solicitudes de conexión entrantes de las aplicaciones que se ejecutan en la modalidad de cliente, consulte "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097.
2. Si no utiliza sus propias colas, ejecute el archivo MQSC de ejemplo amqscos0.tst para crear un conjunto de colas.

   Para hacer esto en sistemas AIX and Linux, especifique:

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

   Compruebe el archivo `sampobj.out` para asegurarse de que no se hayan producido errores.
3. Si desea utilizar las versiones COBOL de los ejemplos Inquire (examinar), Set (establecer) y Echo (eco), cambie las definiciones de proceso antes de ejecutar dichos ejemplos.

   Para los ejemplos Inquire, Set y Echo, las definiciones de ejemplo desencadenan las versiones C de estos ejemplos. Si desea las versiones en COBOL, tendrá que cambiar las definiciones de proceso:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   En Windows, puede hacer esto editando el archivo `amqscos0.tst` y cambiando los nombres de los archivos ejecutables C a los nombres de los archivos ejecutables COBOL antes de utilizar el mandato **runmqsc** para ejecutar estos ejemplos.
4. Ejecute los programas de ejemplo.

   Para ejecutar un ejemplo, especifique su nombre seguido por cualquier parámetro, por ejemplo:

   ```
   amqsput myqueue qmanagername
   ```

   donde *myqueue* es el nombre de la cola en la que se van a colocar los mensajes y *qmanagername* es el gestor de colas propietario de *myqueue*.

   Para obtener más información sobre los parámetros que espera cada uno de los ejemplos, consulte las descripciones de los ejemplos individuales.

   Nota: Para los programas de ejemplo COBOL, cuando pase nombres de cola como parámetros, debe proporcionar 48 caracteres, rellenando con caracteres en blanco si es necesario. Cualquier cantidad distinta a 48 caracteres provocará que el programa falle con código de razón 2085.

Referencia relacionada

"Ejemplos para sistemas AIX and Linux" en la página 1087
Técnicas ilustradas en los programas de ejemplo para IBM MQ for AIX or Linux.

Windows *Preparación y ejecución de programas de ejemplo en Windows*

Antes de ejecutar los programas de ejemplo en Windows, en primer lugar, debe crear un gestor de colas y también crear las colas que necesite. Si desea ejecutar ejemplos de COBOL, es posible que necesite algún tipo de preparación adicional.

Acerca de esta tarea

Los archivos de ejemplo de IBM MQ for Windows se encuentran en los directorios que aparecen en Tabla 161 en la página 1102, si se han utilizado los valores predeterminados en el momento de la instalación. La unidad de la instalación toma de forma predeterminada el valor <c:>.

Tabla 161. Dónde encontrar los ejemplos para IBM MQ for Windows	
Contenido	**Directorio**
Código fuente C	*MQ_INSTALLATION_PATH*\Tools\C\Samples
Código fuente para el ejemplo de manejador de mensajes no entregados	*MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ
Código fuente COBOL	*MQ_INSTALLATION_PATH*\Tools\Cobol \ Ejemplos
Archivos ejecutables C [1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin (versiones de 32 bits) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (versiones de 64 bits)
Archivos MQSC de ejemplo	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Código fuente Visual Basic	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
Ejemplos de .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Ejemplos

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Nota: Hay disponibles versiones de 64 bits de algunos ejemplos de archivos ejecutables C.

Los ejemplos necesitan un conjunto de colas con las que trabajar. Puede utilizar sus propias colas o ejecutar el archivo MQSC de ejemplo amqscos0.tst para crear un conjunto de colas. Para ejecutar los ejemplos, utilice las versiones ejecutables proporcionadas o compile las versiones de código fuente igual que lo haría con cualquier otra aplicación de IBM MQ for Windows.

Procedimiento

1. Cree un gestor de colas y configure las definiciones predeterminadas.

   Debe hacer esto antes de ejecutar cualquiera de los programas de ejemplo. Para obtener más información sobre cómo crear un gestor de colas, consulte Administración de IBM MQ. Para obtener más información sobre cómo configurar un gestor de colas para aceptar de forma segura las solicitudes de conexión entrantes de las aplicaciones que se ejecutan en la modalidad de cliente, consulte “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097.
2. Si no utiliza sus propias colas, ejecute el archivo MQSC de ejemplo amqscos0.tst para crear un conjunto de colas.

   Para hacer esto en sistemas Windows, especifique:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Compruebe el archivo `sampobj.out` para asegurarse de que no se hayan producido errores. Este archivo se encuentra en su directorio actual.
3. Si desea utilizar las versiones COBOL de los ejemplos Inquire (examinar), Set (establecer) y Echo (eco), cambie las definiciones de proceso antes de ejecutar dichos ejemplos.

   Para los ejemplos Inquire, Set y Echo, las definiciones de ejemplo desencadenan las versiones C de estos ejemplos. Si desea las versiones en COBOL, tendrá que cambiar las definiciones de proceso:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

En Windows, puede hacer esto editando el archivo `amqscos0.tst` y cambiando los nombres de los archivos ejecutables C a los nombres de los archivos ejecutables COBOL antes de utilizar el mandato **runmqsc** para ejecutar estos ejemplos.

4. Ejecute los programas de ejemplo.

   Para ejecutar un ejemplo, especifique su nombre seguido por cualquier parámetro, por ejemplo:

   ```
   amqsput myqueue qmanagername
   ```

   donde *myqueue* es el nombre de la cola en la que se van a colocar los mensajes y *qmanagername* es el gestor de colas propietario de *myqueue*.

   Para obtener más información sobre los parámetros que espera cada uno de los ejemplos, consulte las descripciones de los ejemplos individuales.

   Nota: Para los programas de ejemplo COBOL, cuando pase nombres de cola como parámetros, debe proporcionar 48 caracteres, rellenando con caracteres en blanco si es necesario. Cualquier cantidad distinta a 48 caracteres provocará que el programa falle con código de razón 2085.

Referencia relacionada

"Ejemplos para IBM MQ for Windows" en la página 1091
Técnicas ilustradas en los programas de ejemplo para IBM MQ for Windows.

"Ejemplos de Visual Basic para IBM MQ for Windows" en la página 1093
Técnicas ilustradas en los programas de ejemplo de IBM MQ en sistemas Windows.

El programa de ejemplo de salida de API

La salida de API de ejemplo genera un rastreo MQI en un archivo especificado por el usuario con un prefijo definido en la variable de entorno **MQAPI_TRACE_LOGFILE** .

Para obtener más información sobre las salidas de API, consulte "Escritura y compilación de salidas de API en Multiplatforms" en la página 977.

Origen

amqsaxe0.c

Binario

amqsaxe

Configuración para la salida de ejemplo

1. Añada la siguiente información a la sección ApiExitLocal del archivo `qm.ini` .

   Plataformas distintas de Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   donde *MQ_INSTALLATION_PATH* representa el directorio donde IBM MQ está instalado.

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   donde *MQ_INSTALLATION_PATH* representa el directorio donde IBM MQ está instalado.

2. Establecer la variable de entorno **MQAPI_TRACE_LOGFILE**

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. Ejecute la aplicación.

   Los archivos de salida se crean en el directorio `/tmp` con nombres como: `MqiTrace.`*`pid`*`.`*`tid`*`.log`.

Programa de ejemplo de consumo asíncrono

El programa de ejemplo amqscbf ilustra el uso de MQCB y de MQCTL para consumir mensajes de varias colas de forma asíncrona.

amqscbf se proporciona como código fuente C y un cliente binario y un ejecutable de servidor en plataformas AIX, Linux, and Windows .

El programa se inicia por línea de comandos y recibe los siguientes parámetros opcionales:

```
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
```

Proporcione más de un nombre de cola para leer mensajes de varias colas (en el ejemplo se soporta un máximo de diez colas).

Nota: `Reconnect type` sólo es válido para programas cliente.

Ejemplo

El ejemplo muestra amqscbf ejecutando como programa de servidor que lee un mensaje de QL1 y luego se para.

Use IBM MQ Explorer para colocar un mensaje de prueba en QL1. Pare el programa pulsando Intro.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

Qué demuestra amqscbf

El ejemplo muestra cómo leer mensajes de varias colas en el orden de su llegada. Con un MQGET síncrono, esto requeriría mucho más código. En el caso del consumo asíncrono, no es necesario ningún sondeo, y la gestión de hebras y del almacenamiento se realiza mediante IBM MQ. Un ejemplo del "mundo real" tendría que hacer un tratamiento de errores; en el ejemplo, los errores se sacan por consola.

El código de ejemplo sigue los pasos siguientes:

1. Se define la función de devolución de llamada de consumo de mensaje único,

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{ ... }
```

2. Se conecta con el gestor de colas,

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. Se abren las colas de entrada y se asocia cada una de ellas a la función de devolución de llamada `MessageConsumer`.

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   No es necesario configurar `cbd.CallbackFunction` en cada cola; es un campo de solo entrada. No obstante, se podría asociar una función de devolución de llamada distinta a cada cola.

4. Se inicia el consumo de mensajes,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Se espera a que el usuario haya pulsado Intro y luego se para el consumo de mensajes,

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Por último, se desconecta del gestor de colas,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

El programa de ejemplo Asynchronous Put (operación de transferencia asíncrona)

Obtener información sobre el ejemplo amqsapt y el diseño del programa de ejemplo Asynchronous Put.

El programa de ejemplo de operación de transferencia asíncrona coloca mensajes en una cola utilizando la llamada MQPUT asíncrona y luego recupera la información de estado usando la llamada MQSTAT. Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para obtener el nombre de este programa en plataformas diferentes.

Ejecución del ejemplo amqsapt

Este programa acepta hasta 6 parámetros:

1. El nombre de la cola de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Las opciones de apertura (opcional)
4. Las opciones de cierre (opcional)
5. El nombre del gestor de colas de destino (opcional).
6. El nombre de la cola dinámica (optional).

Si no se especifica un gestor de colas, amqsapt se conecta con el gestor de colas predeterminado.

Diseño del programa de ejemplo de colocación asíncrona.

El programa utiliza la llamada MQOPEN con las opciones de salida suministradas, o con las opciones MQOO_OUTPUT y MQOO_FAIL_IF_QUIESCING para abrir la cola de destino para poner mensajes.

Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN. Para que el programa sea sencillo, en esta y en las siguientes llamadas MQI, se utilizarán los valores predeterminados para numerosas opciones.

Para cada línea de entrada, el programa lee el texto en un almacenamiento intermedio y utiliza la llamada MQPUT con MQPMO_ASYNC_RESPONSE para crear un mensaje de datagramas que contenga el texto de

dicha línea y colocarlo asíncronamente en la cola de destino. El programa continúa hasta llegar al final de la entrada o hasta que falla la llamada MQPUT. Si el programa alcanza el final de la entrada, cierra la cola con la llamada MQCLOSE.

A continuación, el programa emite la llamada MQSTAT, devuelve una estructura MQSTS y muestra los mensajes que contienen el número de mensajes colocados correctamente, el número de mensajes colocados con un aviso y el número de anomalías.

Los programas de ejemplo de examen

Los programas de ejemplo de examen examinan los mensajes de una cola utilizando la llamada MQGET.

Consulte "Funciones que se ilustran en los programas de ejemplo en Multiplatforms" en la página 1087 para los nombres de estos programas.

Diseño del programa de ejemplo de examen

El programa abre la cola de destino utilizando la llamada MQOPEN con la opción MQOO_BROWSE. Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN.

Por cada mensaje de la cola, el programa utiliza la llamada MQGET para copiar dicho mensaje de la cola y luego muestra los datos contenidos en el mensaje. La llamada MQGET utiliza estas opciones:

MQGMO_BROWSE_NEXT
Después de la llamada MQOPEN, el cursor para examinar está posicionado lógicamente antes del primer mensaje de la cola, por lo que esta opción hace que se devuelva el ***primer*** cuando se realiza la llamada por primera vez.

MQGMO_NO_WAIT
Si no hay ningún mensaje en la cola, el programa no espera.

MQGMO_ACCEPT_TRUNCATED_MSG
La llamada MQGET especifica un búfer de tamaño fijo. Si un mensaje es más largo que este búfer, el programa mostrará el mensaje truncado junto con un aviso de dicho truncamiento.

El programa muestra cómo debe borrar los campos *MsgId* y *CorrelId* de la estructura MQMD después de cada llamada MQGET, porque la llamada establece estos campos en los valores contenidos en el mensaje que recupera. Si se limpian estos campos, sucesivas llamadas MQGET recuperarán los mensajes en el orden en que estén guardados en la cola.

El programa continúa hasta el final de la cola; la llamada MQGET devuelve el código de razón MQRC_NO_MSG_AVAILABLE y el programa muestra un mensaje de aviso. Si la llamada MQGET falla, el programa muestra un mensaje de error que contiene el código de razón.

A continuación, el programa cierra la cola con la llamada MQCLOSE.

Programas de ejemplo de examen para AIX, Linux, and Windows
Considere utilizar este tema cuando necesite informarse sobre los programas de ejemplo de examen en AIX, Linux, and Windows.

La versión en C del programa recibe dos parámetros:

1. El nombre de la cola de origen (obligatorio).
2. El nombre del gestor de colas (opcional).

Si no se especifica ningún gestor de colas, se conecta con el predeterminado. Por ejemplo, especifique una de las opciones siguientes:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

donde `myqueue` es el nombre de la cola cuyos mensajes se visualizan y `qmanagername` es el gestor de colas que es propietario de `myqueue`.

Si se omite qmanagername, cuando se ejecute el ejemplo en C, este asumirá que el propietario de la cola es el gestor de colas predeterminado.

La versión en COBOL no recibe ningún parámetro. Se conecta con el gestor de colas predeterminado y, cuando se ejecuta, solicita el nombre de la cola de destino:

```
Please enter the name of the target queue
```

Sólo se visualizan los primeros 50 caracteres de cada mensaje, seguidos de `- - - truncated` cuando este es el caso.

Programas de examinar de ejemplo en IBM i
Cada programa recupera copias de todos los mensajes de la cola que especifica cuando llama al programa. Los mensajes permanecen en la cola.

Puede utilizar la cola SYSTEM.SAMPLE.LOCAL proporcionada. Ejecute primero el programa de transferencia de ejemplo para poner algunos mensajes en la cola. Puede utilizar la cola SYSTEM.SAMPLE.ALIAS, que es un nombre de alias para la misma cola local. El programa continúa hasta que llega al final de la cola o hasta que falla la llamada MQI.

Los ejemplos C le permiten especificar el nombre del gestor de colas, generalmente como segundo parámetro, de un modo similar a los ejemplos de los sistemas Windows. Por ejemplo:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Si no se especifica ningún gestor de colas, se conecta con el predeterminado. Esto también es relevante para los ejemplos de RPG. No obstante, en los ejemplos RPG debe proporcionar un nombre de gestor de colas, en lugar de permitir que tome el valor predeterminado.

ALW *El programa de ejemplo del navegador*

El programa de ejemplo del navegador lee y escribe los campos del descriptor de mensaje y los campos de contenido de mensaje de todos los mensajes de una cola.

El programa de ejemplo se escribe como un programa de utilidad, no solo para demostrar una técnica. Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Este programa utiliza estos parámetros posicionales:

1. El nombre de la cola de origen (obligatorio)
2. El nombre del gestor de colas (obligatorio)
3. Un parámetro opcional para las propiedades (opcional)

Utilice las variables de entorno siguientes para proporcionar las credenciales que se utilizan para autenticarse con el gestor de colas:

MQSAMP_USER_ID
Establézcalo en el ID de usuario que se utilizará para la autenticación de conexión, si desea utilizar un ID de usuario y una contraseña para autenticarse con el gestor de colas. El programa solicita la contraseña que acompaña al ID de usuario.

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
Establézcalo en un valor que no esté en blanco si desea proporcionar una señal de autenticación para autenticarse con el gestor de colas. El programa solicita la señal de autenticación. Las señales de autenticación sólo las puede utilizar el ejemplo **amqsbcgc** que utiliza enlaces de cliente.

Para ejecutar estos programas, especifique uno de los mandatos siguientes:

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

donde *myqueue* es el nombre de la cola en la que se van a examinar los mensajes y *qmanagername* es el gestor de colas propietario de *myqueue*.

Lee cada mensaje de la cola y escribe lo siguiente en stdout:

- Campos de descriptor de mensaje formateados
- Datos de mensaje (volcados en hexadecimal y, si es posible, formato carácter)

Tabla 162. Valores permitidos para el parámetro de propiedad

Valor	Comportamiento
0	Comportamiento predeterminado. Las propiedades que se entregan a la aplicación dependen del atributo de cola **PropertyControl** del que se recupera el mensaje.

1	Se crea un manejador de mensajes y se utiliza con MQGET. Las propiedades del mensaje, excepto las contenidas en el descriptor de mensaje (o extensión), se visualizan como en el descriptor de mensaje. Por ejemplo:
	`****Message properties****`
	`  property name: property value`
	O bien, si no hay propiedades disponibles:
	`****Message properties****`
	`  None`
	Los valores numéricos se visualizan utilizando `printf`, los valores de serie se escriben entre comillas simples y las series de bytes se escriben entre X y comillas simples, al igual que en el descriptor de mensaje.

2	Se ha especificado MQGMO_NO_PROPERTIES, por lo que solo se devolverán las propiedades del descriptor de mensaje.
3	Se ha especificado MQGMO_PROPERTIES_FORCE_MQRFH2, por lo que se devuelven todas las propiedades en los datos del mensaje.
4	Se especifica MQGMO_PROPERTIES_COMPATIBILITY, de modo que se pueden devolver todas las propiedades en función de si se incluye una propiedad IBM MQ ; de lo contrario, se descartan las propiedades.

El programa está restringido a la impresión de los primeros 65535 caracteres del mensaje y falla con la razón `truncated msg` si se lee un mensaje más largo.

Para ver un ejemplo de la salida de este programa de utilidad, consulte Examinar colas.

Ejemplo de transacción CICS

Se proporciona un ejemplo de transacción CICS llamado amqscic0.ccs en su versión de código fuente y amqscic0 en su versión ejecutable. Puede crear transacciones usando los recursos estándar de CICS.

Consulte "Creación de una aplicación procedimental" en la página 1025 para obtener detalles sobre los comandos necesarios en su plataforma.

La transacción lee mensajes de la cola de transmisión SYSTEM.SAMPLE.CICS.WORKQUEUE en el gestor de colas predeterminado y los coloca en la cola local, cuyo nombre está contenido en la cabecera de transmisión del mensaje. Las anomalías se envían a la cola SYSTEM.SAMPLE.CICS.DLQ.

Nota: Puede utilizar el script MQSC de ejemplo amqscic0.tst para crear estas colas y las colas de entrada de ejemplo.

El programa de ejemplo de conexión

El programa de ejemplo de conexión permite explorar la llamada MQCONNX y sus opciones en un cliente. El ejemplo se conecta con el gestor de colas con la llamada MQCONNX, consulta el nombre del gestor

de colas con la llamada MQINQ y lo muestra. Además, se proporciona información sobre la ejecución del ejemplo amqscnxc.

Nota: El programa de ejemplo de conexión es un ejemplo de cliente. Puede compilarlo y ejecutarlo en un servidor, pero la función solo tiene sentido para un cliente y solo se suministran archivos ejecutables por el cliente.

Ejecución del ejemplo amqscnxc

La sintaxis de línea de comandos del ejemplo de conexión es:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

Los parámetros son opcionales y su orden no es importante salvo QMgrName, que, si se especifica, tiene que ser el último. Los parámetros son:

ConnName
Nombre de conexión TCP/IP del gestor de colas del servidor

Si no especifica el nombre de conexión TCP/IP, MQCONNX se emite con el valor *ClientConnPtr* establecido a NULL.

SvrconnChannelName
Nombre del canal de conexión del servidor

Si especifica el nombre de la conexión TCP/IP, pero no el canal de conexión del servidor (lo inverso no se permite), el ejemplo utiliza el nombre SYSTEM.DEF.SVRCONN.

Usuario
Nombre de usuario que se va a utilizar en la autenticación de conexión

Si se especifica, el programa solicitará una contraseña que acompañe a ese ID de usuario.

QMgrName
Nombre del gestor de colas de destino

Si no se especifica el gestor de colas de destino, el ejemplo se conecta con cualquier gestor de colas que esté escuchando en el nombre de conexión TCP/IP dado.

Nota: Si especifica un signo de interrogación como único parámetro, o si se especifican parámetros incorrectos, se obtendrá un mensaje que explica cómo utilizar el programa.

Si se ejecuta el ejemplo sin opciones de línea de comandos, se usará el contenido de la variable de entorno MQSERVER para determinar la información de conexión. (En este ejemplo, MQSERVER se establece a `SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com`). Puede ver una salida como esta:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

Si ejecuta el ejemplo y proporciona un nombre de conexión TCP/IP y un nombre de canal de conexión de servidor, pero no un nombre de gestor de colas de destino, como se indica a continuación:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

se utiliza el nombre de gestor de colas predeterminado y aparece una la salida como la siguiente:

```
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
```

Si ejecuta el ejemplo proporcionando un nombre de conexión TCP/IP y un nombre de gestor de colas de destino, como se indica a continuación:

```
amqscnxc -x machine.site.company.com MACHINE
```

verá una salida como esta:

```
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
```

El programa de ejemplo de conexión de datos

El programa de ejemplo de conversión de datos es un esqueleto de una rutina de salida de conversión de datos. Obtenga información sobre el diseño del ejemplo de conversión de datos.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Diseño del ejemplo de conversión de datos

Cada rutina de salida de conversión de datos convierte un solo formato de mensaje con nombre. Este esqueleto está pensado como envoltorio de fragmentos de código generados por la de utilidad de generación de salida de conversión de datos.

La utilidad produce un fragmento de código por cada estructura de datos; varias de dichas estructuras conforman un formato, por lo que se añaden varios fragmentos de código a este esqueleto para generar una rutina que haga la conversión de datos de todo el formato.

A continuación, el programa comprueba si la conversión ha sido satisfactoria o si ha fallado y devuelve los valores necesarios al llamante.

Ejemplos de coordinación de bases de datos

Se proporcionan dos ejemplos que ilustran cómo IBM MQ puede coordinar las actualizaciones de IBM MQ y, también, las actualizaciones de base de datos dentro de la misma unidad de trabajo.

Estos ejemplos son:

1. AMQSXAS0 (en C) o AMQ0XAS0 (en COBOL), que actualiza una sola base de datos dentro de una unidad de trabajo de IBM MQ.
2. AMQSXAG0 (en C) o AMQ0XAG0 (en COBOL), AMQSXAB0 (en C) o AMQ0XAB0 (en COBOL) y AMQSXAF0 (en C) o AMQ0XAF0 (en COBOL), que juntos actualizan dos bases de datos en una unidad de trabajo de IBM MQ, mostrando cómo puede accederse a varias bases de datos. Estos ejemplos se proporcionan para mostrar el uso de la llamada MQBEGIN, las llamadas mixtas SQL y IBM MQ y dónde y cuándo conectarse a una base de datos.

Figura 128 en la página 1111 muestra cómo se utilizan los ejemplos proporcionados para actualizar bases de datos:

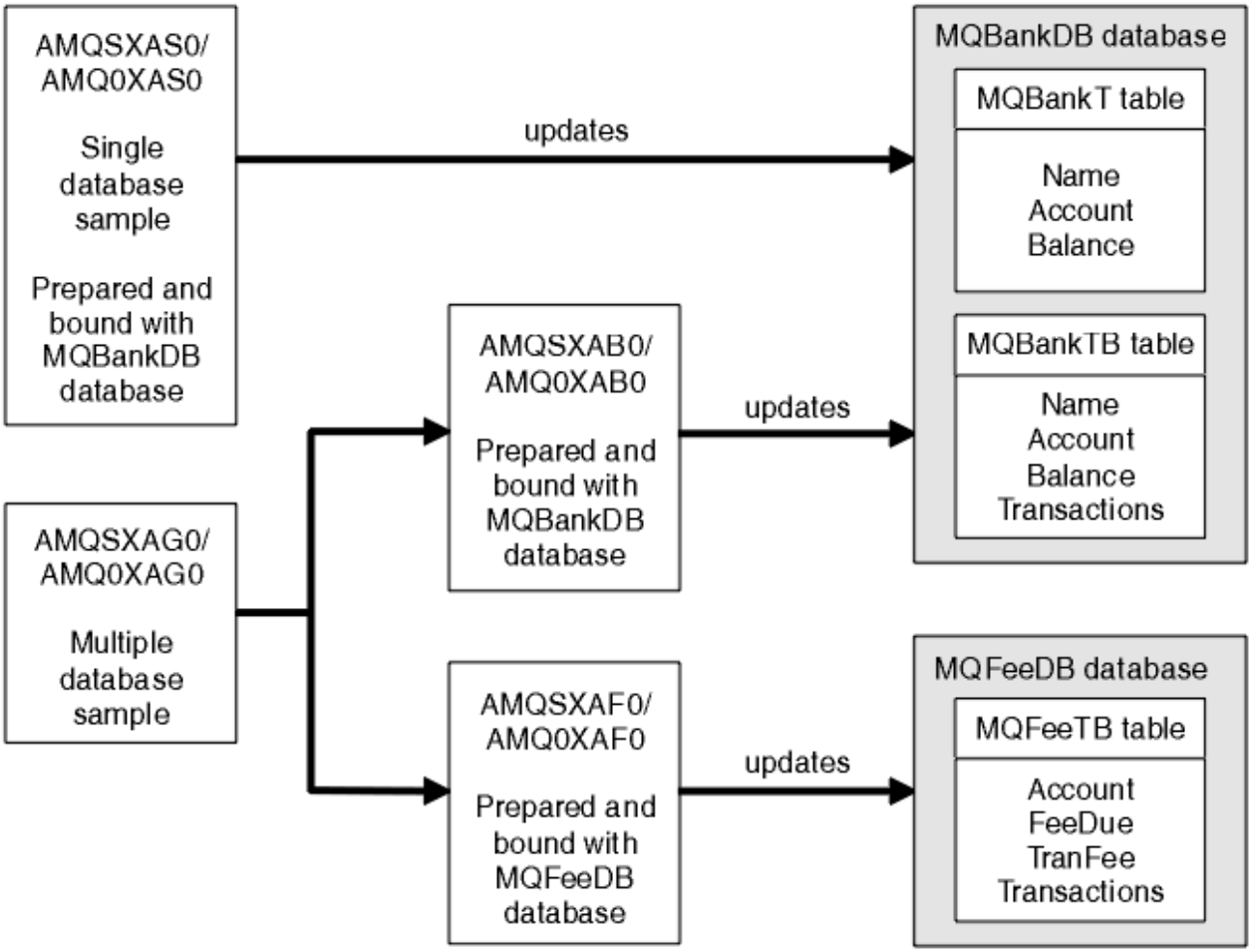


Figura 128. Ejemplos de coordinación de bases de datos

Los programas leen un mensaje de una cola (bajo punto de sincronización) y luego, usando la información del mensaje, obtienen la información relevante de la base de datos y la actualizan. A continuación, se imprime el nuevo estado de la base de datos.

La lógica del programa es la siguiente:

1. Se usa el nombre de la cola de entrada pasada como argumento del programa.
2. Se conecta con el gestor de colas predeterminado (o, de forma opcional, con el nombre proporcionado en C) utilizando MQCONN.
3. Se abre una cola (utilizando MQOPEN) para entrada mientras no haya errores.
4. Se inicia una unidad de trabajo utilizando MQBEGIN.
5. Se obtiene el siguiente mensaje (utilizando MQGET) de la cola bajo el punto de sincronización.
6. Se obtiene la información de las bases de datos.
7. Se actualiza la información de las bases de datos.
8. Se confirman los cambios utilizando MQCMIT.
9. Se imprime la información actualizada (si no hay ningún mensaje disponible, se cuenta como un error y el bucle finaliza).
10. Se cierra la cola utilizando MQCLOSE.
11. Se desconecta de la cola utilizando MQDISC.

En los ejemplos se usan cursores SQL, de modo que las lecturas de las bases de datos (es decir, varias instancias) se bloquean mientras se procesa un mensaje, lo que permite que varias instancias de

estos programas ejecuten simultáneamente. Los cursores se abren de forma explícita, pero se cierran implícitamente con la llamada MQCMIT.

El ejemplo de base de datos única (AMQSXAS0 o AMQ0XAS0) no tiene ninguna sentencia SQL CONNECT y la conexión a la base de datos se realiza implícitamente mediante IBM MQ con la llamada MQBEGIN. El ejemplo de varias bases de datos (AMQSXAG0 o AMQ0XAG0, AMQSXAB0 o AMQ0XAB0 y AMQSXAF0 o AMQ0XAF0) tiene sentencias CONNECT de SQL, ya que algunos productos de base de datos solo permiten una única conexión activa. Si este no es el caso de su producto de base de datos, o si está accediendo a una única base de datos en varios productos de base de datos, se pueden eliminar las sentencias CONNECT de SQL.

Los ejemplos se preparan con el producto de base de datos IBM Db2, por lo que podría tener que modificarlos para que funcionen con otros productos de base de datos.

La comprobación de errores de SQL utiliza las rutinas en UTIL.C y CHECKERR.CBL proporcionadas por Db2. Hay que compilar o sustituir dichas rutinas antes de compilar y enlazar.

Nota: Si se utiliza el código fuente CHECKERR.MFC de COBOL Micro Focus para la comprobación de errores de SQL, hay que pasar el ID de programa a mayúsculas, es decir, CHECKERR, para que AMQ0XAS0 se enlace correctamente.

Creación de bases de datos y tablas
Cree las bases de datos y las tablas antes de compilar los ejemplos.

Para crear las bases de datos, utilice el método habitual para el producto de base de datos, por ejemplo:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Cree las tablas utilizando las sentencias SQL tal como se indica a continuación:

En 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));
```

En 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.
```

Especifique los datos en las tablas utilizando las sentencias SQL tal como se indica a continuación:

```
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);
.
```

Nota: Para COBOL, utilice las mismas sentencias de SQL, pero añada END_EXEC al final de cada línea.

Recompilar, compilar y enlazar los ejemplos
Obtenga información acerca de cómo precompilar, compilar y enlazar los ejemplos en C y COBOL.

Precompile los archivos `.SQC` (en C) y los archivos `.SQB` (en COBOL) y enlácelos con la base de datos adecuada para generar los archivos `.C` o `.CBL`. Para ello, utilice el método habitual para su producto de base de datos.

Precompilación en 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
```

Precompilación en 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
```

Compilación y enlace

Los siguientes mandatos de ejemplo, utilizan símbolos de *DB2TOP* y de *MQ_INSTALLATION_PATH*. *DB2TOP* representa el directorio de instalación del producto Db2. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.

- AIX En AIX, la vía de acceso al directorio es:

```
/usr/lpp/db2_05_00
```

- Windows En los sistemas Windows, la vía de acceso al directorio depende de la vía de acceso que haya elegido durante la instalación del producto. Si elige los valores predeterminados, la vía de acceso es la siguiente:

```
c:\sqllib
```

Nota: Antes emitir el mandato de enlace en los sistemas Windows, asegúrese de que la variable de entorno LIB contiene las vías de acceso a las bibliotecas de Db2 y de IBM MQ.

Copie los archivos siguientes en un directorio temporal:

- El archivo `amqsxag0.c` de su instalación de IBM MQ

  Nota: Este archivo se puede encontrar en los directorios siguientes:

  – Linux AIX En sistemas AIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows En sistemas Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Los archivos `.c` que ha obtenido precompilando los archivos de código fuente `.sqc`, `amqsxas0.sqc`, `amqsxaf0.sqc` y `amqsxab0.sqc`.
- Los archivos `util.c` y `util.h` de su instalación de Db2.

  Nota: Estos archivos se encuentran en el directorio:

  ```
  DB2TOP/samples/c
  ```

Cree los archivos de objetos para cada archivo `.c` utilizando el siguiente mandato del compilador para la plataforma que esté utilizando:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows Sistemas Windows

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

Cree el archivo ejecutable `amqsxag0` utilizando el siguiente mandato de enlace para la plataforma que esté utilizando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Sistemas Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

Cree el archivo ejecutable amqsxas0 utilizando el siguiente mandato de compilación y enlace para la plataforma que esté utilizando:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Sistemas Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Información adicional

AIX Si está trabajando en AIX y desea acceder a Oracle, utilice el compilador xlc_r y enlace con libmqm_r.a.

Ejecución de los ejemplos
Utilice esta información para saber cómo configurar el gestor de colas antes de ejecutar los ejemplos de coordinación de bases de datos en C y COBOL.

Antes de ejecutar los ejemplos, configure el gestor de colas con el producto de base de datos que esté utilizando. Para obtener información sobre cómo hacer esto, consulte Escenario 1: El gestor de colas realiza la coordinación.

Los títulos siguientes proporcionan información sobre cómo ejecutar ejemplos en C y COBOL:

- "Ejemplos en C" en la página 1115
- "Ejemplos en COBOL" en la página 1116

Ejemplos en C

Los mensajes ha de tener el formato siguiente para que se lean de una cola:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT se puede utilizar para colocar los mensajes en la cola.

Los ejemplos de coordinación de bases de datos reciben dos parámetros:

1. Nombre de la cola (obligatorio).
2. Nombre del gestor de colas (opcional).

Suponiendo que se ha creado y configurado un gestor de colas para el ejemplo de base de datos única llamado singDBQM, con una cola llamada singDBQ, se puede incrementar la cuenta del Sr. Fred Bloggs en 50 de la forma siguiente:

```
AMQSPUT singDBQ singDBQM
```

A continuación, teclee el mensaje siguiente:

```
UPDATE Balance change=50 WHERE Account=1
```

Se pueden colocar varios mensajes en la cola.

```
AMQSXAS0 singDBQ singDBQM
```

Luego se imprime el estado actualizado de la cuenta del Sr. Fred Bloggs.

Suponiendo que se ha creado y configurado un gestor de colas para el ejemplo de varias bases de datos llamad multDBQM, con una cola llamada multDBQ, se decrementa la cuenta de la Sra. Mary Brown en 75, como se indica a continuación:

```
AMQSPUT multDBQ multDBQM
```

A continuación, teclee el mensaje siguiente:

```
UPDATE Balance change=-75 WHERE Account=3
```

Se pueden colocar varios mensajes en la cola.

```
AMQSXAG0 multDBQ multDBQM
```

Luego se imprime el estado actualizado de la cuenta de la Sra. Mary Brown.

Ejemplos en COBOL

Los mensajes ha de tener el formato siguiente para que se lean de una cola:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Para simplificar, el `Balance change` (cambio de saldo) tiene que ser un número de ocho caracteres con signo y `Account` (cuenta) tiene que ser un número de ocho caracteres.

El ejemplo AMQSPUT se puede utilizar para colocar los mensajes en la cola.

Los ejemplos no reciben parámetros y utilizan el gestor de colas predeterminado. Se puede configurar para que ejecute solo uno de los ejemplos en cualquier momento. Suponiendo que se ha configurado el gestor de colas predeterminado para el ejemplo de base de datos única, con una cola llamada singDBQ, se puede incrementar la cuenta del Sr. Fred Bloggs en 50 de la forma siguiente:

```
AMQSPUT singDBQ
```

A continuación, teclee el mensaje siguiente:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

Puede colocar varios mensajes en la cola:

```
AMQ0XAS0
```

Escriba el nombre de la cola:

```
singDBQ
```

Luego se imprime el estado actualizado de la cuenta del Sr. Fred Bloggs.

Suponiendo que ha configurado el gestor de colas predeterminado en el ejemplo de varias bases de datos, con una cola llamada multDBQ, la cuenta de la Sra. Mary Brown se decrementa en 75, como se indica a continuación:

```
AMQSPUT multDBQ
```

A continuación, teclee el mensaje siguiente:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

Puede colocar varios mensajes en la cola:

```
AMQ0XAG0
```

Escriba el nombre de la cola:

```
multDBQ
```

Luego se imprime el estado actualizado de la cuenta de la Sra. Mary Brown.

Ejemplo de cola de mensajes no entregados

Se proporciona un manejador de cola de mensajes no entregados de ejemplo, el nombre de la versión ejecutable es `amqsdlq`. Si desea un manejador de cola de mensajes no entregados que sea distinto de **RUNMQDLQ**, el origen del ejemplo está disponible para que lo utilice como base.

El ejemplo es similar al manejador de la cola de mensajes no entregados que se proporciona con el producto pero el rastreo y la notificación de errores son diferentes. Hay dos variables de entorno disponibles:

ODQ_TRACE
: Establézcalo en YES o `yes` para activar el rastreo.

ODQ_MSG
: Establézcalo en el nombre del archivo que contiene los mensajes de error y de información. El archivo proporcionado se denomina `amqsdlq.msg`.

Debe hacer que el entorno reconozca estas variables con los mandatos **`export`** o **`set`**, en función de su plataforma. El rastreo se desactiva con el mandato **`unset`**.

Puede modificar el archivo de mensajes de error, `amqsdlq.msg`, para que se ajuste a sus propios requisitos. El ejemplo coloca los mensajes en stdout, y **no** en el archivo de registro de errores de IBM MQ.

Para obtener más información sobre cómo funciona el manejador de mensajes no entregados y cómo lo ejecuta, consulte Proceso de mensajes en una cola de mensajes no entregados de IBM MQ o la *Guía de gestión del sistema* para su plataforma.

El programa de ejemplo de lista de distribución

El ejemplo de lista de distribución amqsptltl0 ilustra cómo colocar un mensaje en varias colas de mensajes. Se basa en el ejemplo de MQPUT, amqsput0.

Ejecución del ejemplo de lista de distribución amqsptl0

El ejemplo de lista de distribución se ejecuta de forma similar a los ejemplos de colocación.

Recibe los parámetros siguientes:

- Los nombres de las colas.
- Los nombres de los gestores de colas

Estos valores se especifican como pares. Por ejemplo:

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

Las colas se abren con MQOPEN y los mensajes se colocan en las colas utilizando MQPUT. Se devuelven códigos de razón si no se reconoce alguno de los nombres de cola o de gestor de colas.

No olvide definir los canales entre gestores de colas para que los mensajes puedan fluir entre ellos. El programa de ejemplo no lo hace por usted.

Diseño del ejemplo de lista de distribución

Un registro de colocación de mensaje (MQPMR) especifica los atributos de mensaje de cada destino. El ejemplo proporciona los valores de *MsgId* y *CorrelId*, y estos sustituyen los valores especificados en la estructura MQMD.

El campo *PutMsgRecFields* de la estructura MQPMO indica qué campos están presentes en los MQPMR:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

A continuación, el ejemplo asigna los registros de respuesta y los de objeto. Un registro de objeto (MQOR) requiere al menos un par de nombres y un número par de nombres, es decir, *ObjectName* y *ObjectQMgrName*.

La siguiente etapa implica conectar con los gestores de colas utilizando MQCONN. El ejemplo intenta conectar con el gestor de colas asociado a la primera cola en el MQOR; si esto falla, recorre los registros de objeto uno a uno. Recibirá una notificación si no fuera posible conectar con ningún gestor de colas y saliera el programa.

Las colas de destino se abren utilizando MQOPEN y el mensaje se coloca en estas colas utilizando MQPUT. Se informa de cualquier problema y error en los registros de respuestas (MQRR).

Por último, las colas de destino se cierran utilizando MQCLOSE y el programa se desconecta del gestor de colas utilizando MQDISC. Se usan los mismos registros de respuesta en cada llamada indicando *CompCode* y *Reason*.

Los programas de ejemplo de eco

Los programas de ejemplo de eco hacen eco de un mensaje de una cola de mensajes a la cola de respuestas.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Los programas están pensados para ejecutarse como programas desencadenados.

En sistemas IBM i, AIX, Linux, and Windows, su única entrada es una estructura MQTMC2 (mensaje desencadenante) que contiene el nombre de una cola de destino y del gestor de colas. La versión en COBOL utiliza el gestor de colas predeterminado.

IBM i En IBM i, para que el proceso de desencadenamiento funcione, asegúrese de que el programa de ejemplo Echo que desea utilizar esté desencadenado por los mensajes que llegan a la cola SYSTEM.SAMPLE.ECHO. Para ello, especifique el nombre del programa de ejemplo Echo que desea utilizar en el campo *ApplId* de la definición de proceso SYSTEM.SAMPLE.ECHOPROCESS. (Para ello, puede utilizar el comando CHGMQMPRC; para obtener detalles, consulte Cambiar proceso MQ (CHGMQMPRC)). La cola de ejemplo tiene un tipo de desencadenante FIRST, por tanto, si ya hay mensajes en la cola antes de ejecutar el ejemplo de solicitud, los mensajes que envíe no desencadenarán el ejemplo de eco.

Cuando haya establecido correctamente la definición, primero inicie AMQSERV4 en un trabajo y luego inicie AMQSREQ4 en otro. Puede utilizar AMQSTRG4, en lugar de AMQSERV4 pero, debido a que pueden producirse retardos en el envío de trabajos, es posible que no le resulte tan fácil seguir lo que está sucediendo.

Utilice los programas de ejemplo de petición para enviar mensajes a la cola SYSTEM.SAMPLE.ECHO. Los programas de ejemplo de eco envían un mensaje de respuesta que contiene los datos del mensaje de respuesta a la cola de respuestas especificada en el mensaje de petición.

Diseño de los programas de ejemplo de eco

El programa abre la cola referenciada en la estructura de mensajes de desencadenante que se le pasó al iniciarlo. (A efectos de claridad, nos referiremos a esta cola como cola de peticiones). El programa usa la llamada MQOPEN para abrir esta cola para entrada compartida.

El programa utiliza la llamada MQGET para eliminar mensajes de esta cola. En esta llamada se utilizan las opciones MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT y MQGMO_WAIT, con un intervalo de espera de 5 segundos. El programa comprueba el descriptor de cada mensaje para ver si es un mensaje de solicitud; si no lo es, descarta el mensaje y muestra un mensaje de advertencia.

Por cada línea de entrada, el programa lee el texto en un búfer y utiliza la llamada MQPUT1 para colocar un mensaje de solicitud, que contiene el texto de dicha línea, en la cola de respuestas.

Si la llamada MQGET falla, el programa coloca un mensaje de informe en la cola de respuesta, estableciendo el campo *Feedback* del descriptor de mensaje en el código de razón devuelto por MQGET.

Cuando no quedan mensajes en la cola de solicitudes, el programa cierra esa cola y se desconecta del gestor de colas.

IBM i En IBM i, el programa también puede responder a los mensajes enviados a la cola de plataformas distintas a IBM MQ for IBM i, aunque no se proporciona ningún ejemplo para esta situación. Para hacer que funcione el programa ECHO:

- Escriba un programa especificando correctamente los parámetros **Format**, **Encoding** y **CCSID**, para enviar mensajes de solicitud de texto.

  El programa ECHO solicita al gestor de colas que realice la conversión de datos del mensaje, si es necesario.
- Especifique CONVERT(*YES) en el canal IBM MQ for IBM i emisor si el programa que ha escrito no proporciona una conversión similar para la respuesta.

Programas de ejemplo de obtención

Los programas de ejemplo de obtención obtienen los mensajes de una cola utilizando la llamada MQGET.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Diseño del programa de ejemplo de obtención

El programa abre la cola de destino utilizando la llamada MQOPEN con la opción MQOO_INPUT_AS_Q_DEF. Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN.

Por cada mensaje de la cola, el programa utiliza la llamada MQGET para eliminar dicho mensaje de la cola y luego muestra los datos contenidos en el mensaje. La llamada MQGET utiliza la opción MQGMO_WAIT, especificando un *WaitInterval* de 15 segundos, para que el programa espere este periodo si no hay ningún mensaje en la cola. Si no llega ningún mensaje antes de que venza este intervalo, la llamada falla y devuelve el código de razón MQRC_NO_MSG_AVAILABLE.

El programa muestra cómo debe borrar los campos *MsgId* y *CorrelId* de la estructura MQMD después de cada llamada MQGET porque la llamada establece estos campos en los valores contenidos en el mensaje que recupera. Si se limpian estos campos, sucesivas llamadas MQGET recuperarán los mensajes en el orden en que estén guardados en la cola.

La llamada MQGET especifica un búfer de tamaño fijo. Si un mensaje es más largo que este búfer, la llamada falla y el programa se para.

El programa continúa hasta que la llamada MQGET devuelve el código de razón MQRC_NO_MSG_AVAILABLE o falla. Si la llamada falla, el programa muestra un mensaje de error que contiene el código de razón.

A continuación, el programa cierra la cola con la llamada MQCLOSE.

Ejecución de los ejemplos amqsget y amqsgetc

Estos programas reciben los siguientes parámetros posicionales:

1. El nombre de la cola de origen (obligatorio)
2. El nombre del gestor de colas (opcional).

   Si no se especifica un gestor de colas, **amqsget** se conecta al gestor de colas predeterminado y **amqsgetc** se conecta al gestor de colas identificado por la variable de entorno *MQSERVER* o el archivo de definición de canal de cliente.
3. Opciones de apertura (opcional).

   Si no se especifican las opciones de apertura, el ejemplo utiliza el valor 8193, que es la combinación de estas dos opciones:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. Opciones de cierre (opcional).

   Si no se especifican opciones de cierre, el ejemplo utiliza el valor 0, que es MQCO_NONE.

Utilice las variables de entorno siguientes para proporcionar las credenciales que se utilizan para autenticarse con el gestor de colas:

MQSAMP_USER_ID
Establézcalo en el ID de usuario que se utilizará para la autenticación de conexión, si desea utilizar un ID de usuario y una contraseña para autenticarse con el gestor de colas. El programa solicita la contraseña que acompaña al ID de usuario.

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
Establézcalo en un valor que no esté en blanco si desea proporcionar una señal de autenticación para autenticarse con el gestor de colas. El programa solicita la señal de autenticación. Las señales de autenticación sólo las puede utilizar el ejemplo **amqsgetc** que utiliza enlaces de cliente.

Para ejecutar estos programas, especifique una de las opciones siguientes:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

donde *myqueue* es el nombre de la cola de la que el programa va a obtener los mensajes y *qmanagername* es el gestor de colas que es propietario de *myqueue*.

Utilización de amqsget y amqsgetc

Tenga en cuenta que **amqsget** realiza una conexión local al gestor de colas, utilizando memoria compartida para conectarse al gestor de colas y, como tal, solo se se puede ejecutar en el sistema en el que reside el gestor de colas, mientras que **amqsgetc** realiza una conexión de estilo cliente (incluso si se está conectando a un gestor de colas en el mismo sistema).

Cuando se utiliza **amqsgetc**, tendrá que proporciona los detalles de la aplicación sobre cómo accede actualmente al gestor de colas, en términos de host o dirección IP del gestor de colas y el puerto de escucha del gestor de colas.

Normalmente, esto se realiza utilizando la variable de entorno **MQSERVER** o definiendo detalles de conexión utilizando una tabla de definición de canal de cliente, que también se puede proporcionar a **amqsgetc** utilizando variables de entorno; por ejemplo, consulte MQCCDTURL.

Un ejemplo que utiliza **MQSERVER**, que se conecta a un gestor de colas localmente, que tiene un escucha que se ejecuta en el puerto 1414 y que utiliza el canal de conexión de servidor predeterminado es:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

Programas de ejemplo de alta disponibilidad

Los programas de ejemplo de alta disponibilidad **amqsghac**, **amqsphac** y **amqsmhac** utilizan la reconexión automática de cliente para comprobar la recuperación después de un error de un gestor de colas. **amqsfhac** comprueba que un gestor de colas que utiliza almacenamiento en red mantiene integridad de datos tras una anomalía.

Los programas **amqsghac**, **amqsphac** y **amqsmhac** se inician desde la línea de mandatos y se pueden utilizar conjuntamente para comprobar la reconexión después de un error de un gestor de colas de varias instancias.

Como alternativa, también puede utilizar los programas **amqsghac**, **amqsphac** y **amqsmhac** para comprobar la reconexión de un cliente a gestores de colas de una sola instancia, configurado normalmente en un grupo de gestores de colas.

Para que el ejemplo sea sencillo y fácil de configurar, se muestran los programas de ejemplo que se reconectan a un gestor de colas de una sola instancia que se ha iniciado, detenido y luego se ha reiniciado; consulte “Configurar y controlar el gestor de colas” en la página 1123.

Utilice **amqsfhac** en paralelo con **amqmfsck** para comprobar la integridad del sistema de archivos. Consulte **amqmfsck** (comprobación del sistema de archivos) y Verificación del comportamiento del sistema de archivos compartidos para obtener más información.

amqsphac *nombre_cola* [*nombre_gestor_colas*]

- **amqsphac** es una aplicación de IBM MQ MQI client. Transfiere una secuencia de mensajes a una cola con un retardo de dos segundos entre cada mensaje y visualiza sucesos enviados al gestor de sucesos.
- No se utiliza ningún punto de sincronismo para transferir mensajes a la cola.
- La reconexión se puede realizar en cualquier gestor de colas del mismo grupo de gestores de colas.

amqsghac *nombre_cola* [*nombre_gestor_colas*]

- **amqsghac** es una aplicación de IBM MQ MQI client. Obtiene mensajes de una cola y visualiza los sucesos que se envían a su gestor de sucesos.
- No se utiliza ningún punto de sincronización para obtener mensajes de la cola.
- La reconexión se puede realizar en cualquier gestor de colas del mismo grupo de gestores de colas.

amqsmhac -s *NombreColaOrigen* -t *NombreColaDestino* [-m *nombre_gestor_colas]*] [-w *Intervalo_espera*]

- **amqsmhac** es una aplicación de IBM MQ MQI client. Copia mensajes de una cola a otra con un intervalo de espera predeterminado de 15 minutos después del último mensaje que se ha recibido antes de que finalice el programa.
- Los mensajes se copian dentro del punto de sincronización.
- La reconexión sólo se puede realizar en el mismo gestor de colas.

amqsfhac *NombreGestorColas NombreCola NombreColaAuxiliar RecuentoTransacciones RecuentoRepeticiones* (0 | 1 | 2)

- **amqsfhac** es una aplicación de IBM MQ MQI client. Comprueba que un gestor de colas de varias instancias de IBM MQ que utiliza almacenamiento en red, tal como un NAS o un sistema de archivos de clúster, mantiene la integridad de los datos. Siga los pasos para ejecutar **amqsfhac** en Verificación del comportamiento del sistema de archivos compartidos.
- Utiliza la opción MQCNO_RECONNECT_Q_MGR al conectar con *QueueManagerName*. Se reconecta automáticamente cuando el gestor de colas falla.

- Transfiere *RecuentoTransacciones*RecuentoRepeticiones* mensajes persistentes a *GestorColas* durante el tiempo que ha hecho que el gestor de colas falle varias veces. **amqsfhac** se vuelve a conectar al gestor de colas cada vez y continúa. La prueba es para asegurarse de que no se pierda ningún mensaje.
- Dentro de cada transacción se transfieren *RecuentoTransacciones* mensajes. La transacción se repite el número de veces indicado por *RecuentoRepeticiones*. Si se produce un error dentro de una transacción, **amqsfhac** se retrotrae y vuelve a someter la transacción cuando **amqsfhac** se reconecta al gestor de colas.
- También transfiere mensajes a *NombreColaAuxiliar*. *NombreColaAuxiliar* se utiliza para comprobar si todos los mensajes de *NombreCola* se confirman o retrotraen correctamente. Si detecta una incoherencia, escribe un mensaje de error.
- Puede modificar el nivel de rastreo de la salida desde **amqsfhac** estableciendo el último parámetro en `(0|1|2)`.

  0
  Salida mínima.

  1
  Salida media.

  2
  Salida máxima.

Configuración de una conexión de cliente

Debe configurar un canal de conexión de cliente y servidor para ejecutar los ejemplos. El procedimiento de verificación de cliente describe cómo configurar un entorno de prueba de cliente.

Como alternativa, utilice la configuración proporcionada en el ejemplo siguiente.

Ejemplo de uso de amqsghac, amqsphac y amqsmhac

Este ejemplo muestra los clientes reconectables utilizando un gestor de colas de una sola instancia.

Los mensajes se colocan en la cola SOURCE mediante **amqsphac**, se transfieren a TARGET mediante **amqsmhac**, y se recuperan de TARGET mediante **amqsghac**; consulte Figura 129 en la página 1122.

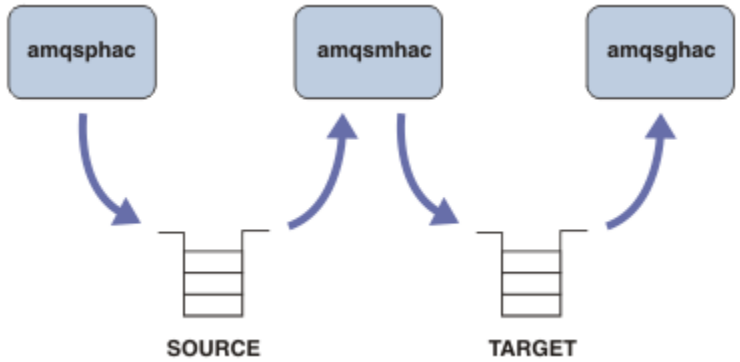


Figura 129. Ejemplos de cliente reconectable

Siga estos pasos para ejecutar los ejemplos.

1. Cree un archivo `hasamples.tst` que contenga los mandatos:

```
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. Escriba los mandatos siguientes en un indicador de mandatos:
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. Establezca la variable de entorno **MQCHLLIB** en la vía de acceso al archivo de definición de canal de cliente AMQCLCHL.TAB ; por ejemplo, `SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Abra tres ventanas nuevas con **MQCHLLIB** definido; por ejemplo, en Windows, escriba **start** tres veces en el indicador de mandatos anterior iniciando cada uno de los programas en una de las ventanas. Consulte el paso "5" en la página 1124 en "Configurar y controlar el gestor de colas" en la página 1123.
5. Escriba el mandato `endmqm -r -p QM1` para detener el gestor de colas y luego permita que los clientes se reconecten.
6. Escriba el mandato `strmqm QM1` para reiniciar el gestor de colas.

Los resultados de la ejecución de **amqsghac**, **amqsphac** y **amqsmhac** en Windows se muestran en los ejemplos siguientes.

Configurar y controlar el gestor de colas

1. Cree el gestor de colas.

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Recuerde el directorio de datos para establecer la variable **MQCHLLIB** más adelante.

2. Inicie el gestor de colas.

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. Cree las colas y canales, modifique el puerto de escucha, e inicie el escucha y el canal.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
```

```
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Dé a conocer la tabla de canales cliente a los clientes.

   Utilice el directorio de datos devuelto por el mandato **crtmqm** en el paso “1” en la página 1123, y agréguele el directorio `@ipcc` para definir la variable **MQCHLLIB**.

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Inicie los programas de ejemplo en las demás ventanas

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Finalice el gestor de colas y reinícielo de nuevo.

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
```

```
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

Tareas relacionadas

Verificación del comportamiento del sistema de archivos compartidos

Referencia relacionada

amqmfsck (comprobación del sistema de archivos)

Programas de ejemplo de consulta

Los programas de ejemplo de consulta consultan algunos de los atributos de una cola con la llamada MQINQ.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Estos programas están pensados para ejecutarse como programas desencadenados, por lo que su única entrada es una estructura MQTMC2 (mensaje desencadenante) para IBM MQ for Multiplatforms. Esta estructura contiene el nombre de la cola de destino cuyos atributos se van a consultar. La versión en C también utiliza el nombre del gestor de colas. La versión en COBOL utiliza el gestor de colas predeterminado.

Para que el proceso de desencadenamiento funcione, asegúrese de que el programa de ejemplo de consulta que desee utilizar esté desencadenado por los mensajes que lleguen a la cola SYSTEM.SAMPLE.INQ. Para ello, especifique el nombre del programa de ejemplo de consulta que desea utilizar en el campo *ApplicId* de la definición de proceso SYSTEM.SAMPLE.INQPROCESS.

IBM i Para IBM i, puede utilizar el mandato CHGMQMPRC para esto; para obtener detalles, consulte Cambiar proceso MQ (CHGMQMPRC). La cola de ejemplo tiene un tipo de desencadenante FIRST; si ya hay mensajes en la cola antes de ejecutar el ejemplo de solicitud, el ejemplo de consulta no se desencadenará por los mensajes que le envíe.

Cuando haya configurado correctamente la definición:

- ALW Para AIX, Linux, and Windows, inicie el programa **runmqtrm** en una sesión y, a continuación, inicie el programa `amqsreq` en otra.
- IBM i En IBM i, inicie el programa AMQSERV4 en una sesión y, a continuación, inicie el programa AMQSREQ4 en otra sesión. Puede utilizar AMQSTRG4, en lugar de AMQSERV4 pero, debido a que pueden producirse retardos en el envío de trabajos, es posible que no le resulte tan fácil seguir lo que está sucediendo.

Utilice los programas de ejemplo de solicitud para enviar mensajes de solicitud, cada uno de los cuales solo contiene un nombre de cola, a la cola SYSTEM.SAMPLE.INQ. Por cada mensaje de solicitud, los programas de ejemplo de consulta envían un mensaje de respuesta que contiene información sobre la cola especificada en el mensaje de solicitud. Las respuestas se envían a la cola de respuestas especificada en el mensaje de solicitud.

IBM i En IBM i, si se utiliza el miembro del archivo de entrada de ejemplo QMQMSAMP.AMQSDATA(INQ), la última cola nombrada no existe, así que el ejemplo devuelve un mensaje de informe con un código de razón para la anomalía.

Diseño del programa de ejemplo de consulta

El programa abre la cola referenciada en la estructura de mensajes de desencadenante que se le pasó al iniciarlo. (Para que quede más claro, esta cola se llamará *cola de solicitudes*). El programa usa la llamada MQOPEN para abrir esta cola para entrada compartida.

El programa utiliza la llamada MQGET para eliminar mensajes de esta cola. En esta llamada se utilizan las opciones MQGMO_ACCEPT_TRUNCATED_MSG y MQGMO_WAIT con un intervalo de espera de 5 segundos. El programa comprueba el descriptor de cada mensaje para ver si es un mensaje de solicitud; si no lo es, descarta el mensaje y muestra un mensaje de advertencia.

Por cada mensaje de solicitud eliminado de la cola de solicitudes, el programa lee el nombre de la cola (que llamaremos *cola de destino*) contenida en los datos y abre dicha cola usando la llamada MQOPEN con la opción MQOO_INQ. Después, el programa utiliza la llamada MQINQ para consultar los valores de los atributos *InhibitGet*, **CurrentQDepth** y **OpenInputCount** de la cola de destino.

Si la llamada MQINQ es satisfactoria, el programa utiliza la llamada MQPUT1 para colocar un mensaje de respuesta en la cola de respuestas. Este mensaje contiene los valores de los tres atributos.

Si la llamada MQOPEN o MQINQ no es satisfactoria, el programa utiliza la llamada MQPUT1 para colocar un mensaje de informe en la cola de respuestas. En el campo *Feedback* del descriptor de mensaje de este mensaje de informe se encuentra el código de razón devuelto por la llamada MQOPEN o MQINQ, en función de cuál haya fallado.

Después de la llamada MQINQ, el programa cierra la cola de destino con la llamada MQCLOSE.

Cuando no quedan mensajes en la cola de solicitudes, el programa cierra esa cola y se desconecta del gestor de colas.

Programa de ejemplo de consulta de propiedades de un manejador de mensajes

AMQSIQMA es un ejemplo de programa en C para consultar las propiedades de un manejador de mensajes de una cola de mensajes y ejemplifica el uso de la llamada de API MQINQMP.

Este ejemplo crea un descriptor de mensaje y lo coloca en el campo MsgHandle de la estructura MQGMO. A continuación, el ejemplo obtiene un mensaje y consulta e imprime todas las propiedades con las que se ha llenado el descriptor de mensaje.

```
C:\Archivos de programa\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

Los programas de ejemplo de publicación/suscripción

Los programas de ejemplo de publicación/suscripción muestran el uso de las características de publicación y suscripción en IBM MQ.

Hay tres programas de ejemplo en lenguaje C que ilustran cómo programar en la interfaz de publicación/suscripción de IBM MQ. Hay algunos ejemplos en C que utilizan las interfaces más antiguas y hay ejemplos Java. Los ejemplos Java utilizan la interfaz de publicación/suscripción de IBM MQ en com.ibm.mq.jar y la interfaz de publicación/suscripción de JMS en com.ibm.mqjms. Los ejemplos de JMS no se tratan en este tema.

C

Busque el ejemplo de publicador `amqspub` en la carpeta de ejemplos en lenguaje C. Ejecútelo con un nombre de tema cualquiera como primer parámetro, seguido de un nombre de gestor de colas opcional. Por ejemplo, `amqspub mytopic QM3`. También existe una versión de cliente llamada `amqspubc`. Si elige ejecutar la versión cliente, consulte primero “Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms” en la página 1097 para obtener más detalles.

El publicador se conecta al gestor de colas predeterminado y responde con la salida, `target topic is mytopic` . Cada línea que especifique en esta ventana se publicará a partir de ahora en `mytopic`.

Abra otra ventana de comandos en el mismo directorio y ejecute el programa suscriptor, `amqssub`, proporcionándole el mismo nombre de tema y un nombre de gestor de colas opcional. Por ejemplo, `amqssub mytopic QM3`.

El suscriptor responde con la salida `Calling MQGET : 30 seconds wait time` . A partir de ahora, las líneas que escriba en el editor aparecerán en la salida del suscriptor.

Inicie otro suscriptor en otra ventana de comandos y observe cómo ambos suscriptores reciben las publicaciones.

Para obtener la documentación completa de los parámetros, incluidas las opciones de configuración, consulte el código fuente del ejemplo. Los valores del campo de opciones de suscriptor se describen en el tema siguiente: Opciones (MQLONG).

Hay otro ejemplo de suscriptor, `amqssbx`, que ofrece opciones de suscripción adicionales en forma de conmutadores por línea de comandos.

Escriba `amqssbx -d mysub -t mytopic -k` para invocar al suscriptor utilizando suscripciones duraderas que se conservan después de que el suscriptor haya terminado.

Pruebe la suscripción publicando otro elemento con el publicador. Espere 30 segundos para que el suscriptor finalice. Publique algunos elementos más bajo el mismo tema. Reinicie el suscriptor. Inmediatamente después de reiniciar el suscriptor, se muestra el último elemento publicado mientras no ejecutaba el suscriptor.

Legado C

Existe un conjunto adicional de ejemplos en lenguaje C que ilustran los comandos encolados. Algunos de estos ejemplos se proporcionaron originalmente como parte de MQ0C Supportpac. Por motivos de compatibilidad, las funciones mostradas en los ejemplos están plenamente soportadas.

No le recomendamos usar la interfaz de comandos encolados. Es mucho más compleja que el API de publicación/suscripción y no existe ninguna razón funcional convincente para programar complejos comandos encolados. No obstante, puede que el enfoque de encolamiento le resulte más adecuado, quizás porque ya esté utilizando la interfaz o porque su entorno de programación facilite crear un mensaje complejo y llamar a una MQPUT genérica, en lugar de construir diferentes llamadas a MQSUB.

Los ejemplos adicionales se encuentran en el subdirectorio `pubsub` de la carpeta `samples`.

En Tabla 163 en la página 1127 se muestran seis tipos de ejemplo.

Tabla 163. Categorías de programas de ejemplo de publicación/suscripción en C legados

Categoría	Programas	Comentarios
RFH1	`amqssr1a.c` `amqspr1a.c`	Ejemplo sencillo de publicación/suscripción que usa mensajes en formato RFH1.
RFH2	`amqssr2a.c` `amqspr2a.c`	Ejemplo sencillo de publicación/suscripción que usa mensajes en formato RFH2.
Ejemplos de MQAI	`amqsppca.c` `amqsspca.c`	Ejemplo sencillo de publicación/suscripción creado mediante comandos PCF y la interfaz de comandos MQAI.
Servicio de resultados MA0C utilizando RFH1	`amqsgama.c` `amqsresa.c`	Servicio de resultados utilizando cabeceras RFH1 1. Requiere las colas definidas en `amqsgama.tst` y `amqsresa.tst` 2. `amqsresa` se debe iniciar antes de `amqsgama`

Tabla 163. Categorías de programas de ejemplo de publicación/suscripción en C legados (continuación)		
Categoría	**Programas**	**Comentarios**
Servicio de resultados MA0C utilizando RFH2	`amqsgr2a.c` `amqsrr2a.c`	Servicio de resultados utilizando cabeceras RFH2 1. Requiere las colas definidas en `amqsgama.tst` y `amqsresa.tst` 2. `amqsresa` se debe iniciar antes de `amqsgama`
Ejemplo de publicación/ suscripción de salida de direccionamiento	`amqspsra.c`	Ilustra cómo cambiar el destino de la cola o del gestor de colas de un mensaje de publicación/suscripción en una salida de direccionamiento.

Programa de ejemplo para Java

El ejemplo de Java `MQPubSubApiSample.java` combina publicadores y suscriptores en un solo programa. Sus archivos fuente y sus archivos de clase compilados se encuentran en la carpeta de ejemplos `wmqjava`.

Si opta por ejecutar en modo cliente, consulte primero "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097 para obtener más detalles.

Ejecute el ejemplo desde la línea de mandatos utilizando el mandato Java, si tiene un entorno Java configurado. También puede ejecutar el ejemplo desde el espacio de trabajo de IBM MQ Explorer Eclipse que tiene un entorno de trabajo de programación Java ya configurado.

Puede que tenga que cambiar algunas de las propiedades del programa de ejemplo para poder ejecutarlo. Para ello, proporcione los parámetros pertinentes a la JVM, o edite el código fuente.

Las instrucciones en "Ejecución del ejemplo Java MQPubSubApiSample" en la página 1128 muestran cómo ejecutar el ejemplo en el espacio de trabajo de Eclipse.

Ejecución del ejemplo Java MQPubSubApiSample
Cómo ejecutar el ejemplo MQPubSubApiSample utilizando las herramientas de desarrollo Java desde la plataforma Eclipse.

Antes de empezar

Abra el entorno de trabajo Eclipse. Cree un nuevo directorio de espacio de trabajo y selecciónelo. Cierre la ventana de bienvenida.

Siga los pasos de "Configuración de un gestor de colas para aceptar conexiones de cliente en Multiplatforms" en la página 1097 antes de ejecutar como cliente.

Acerca de esta tarea

El programa de ejemplo de publicación/suscripción de Java es un programa IBM MQ MQI client Java. El ejemplo se ejecuta sin modificaciones utilizando un gestor de colas predeterminado que está en escucha en el puerto 1414. La tarea describe este caso simple e indica en términos generales cómo proporcionar parámetros y modificar el ejemplo para ajustarlo a las distintas configuraciones de IBM MQ. El ejemplo se ilustra con la ejecución en Windows. Las vías de acceso de archivo serán diferentes en otras plataformas.

Procedimiento

1. Importe los programas Java de ejemplo
   a) En el entorno de trabajo, pulse **Ventana** > **Abrir perspectiva** > **Otras** > **Java** y pulse **Aceptar**.
   b) Vaya a la vista del **Explorador de paquetes**.
   c) Pulse el botón derecho del ratón sobre el espacio en blanco de la vista del **Explorador de paquetes**. Pulse **Nuevo** > **Proyecto Java**.
   d) En el campo **`Project name`** , escriba `MQ Java Samples`. Pulse **Siguiente**.
   e) En el panel **`Java Settings`** , cambie a la pestaña **Bibliotecas** .
   f) Pulse **Añadir JAR externos**.
   g) Vaya a *MQ_INSTALLATION_PATH* `\java\lib` donde *MQ_INSTALLATION_PATH* es la carpeta de instalación de IBM MQ y seleccione `com.ibm.mq.jar` y `com.ibm.mq.jmqi.jar`
   h) Pulse **Abrir** > **Finalizar**.
   i) Pulse el botón derecho del ratón sobre `src` en la vista del **Explorador de paquetes**.
   j) Seleccione **Importar ...** > **General** > **Sistema de archivos** > **Siguiente** > **Examinar**... y vaya a la vía de acceso *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ .
   k) En el panel **Importar**, Figura 130 en la página 1130, pulse `samples` (no marque el recuadro de selección).
   l) Seleccione `MQPubSubApiSample.java`. El campo **`Into folder`** debe contener `MQ Java Samples/src`. Pulse **Finalizar**.

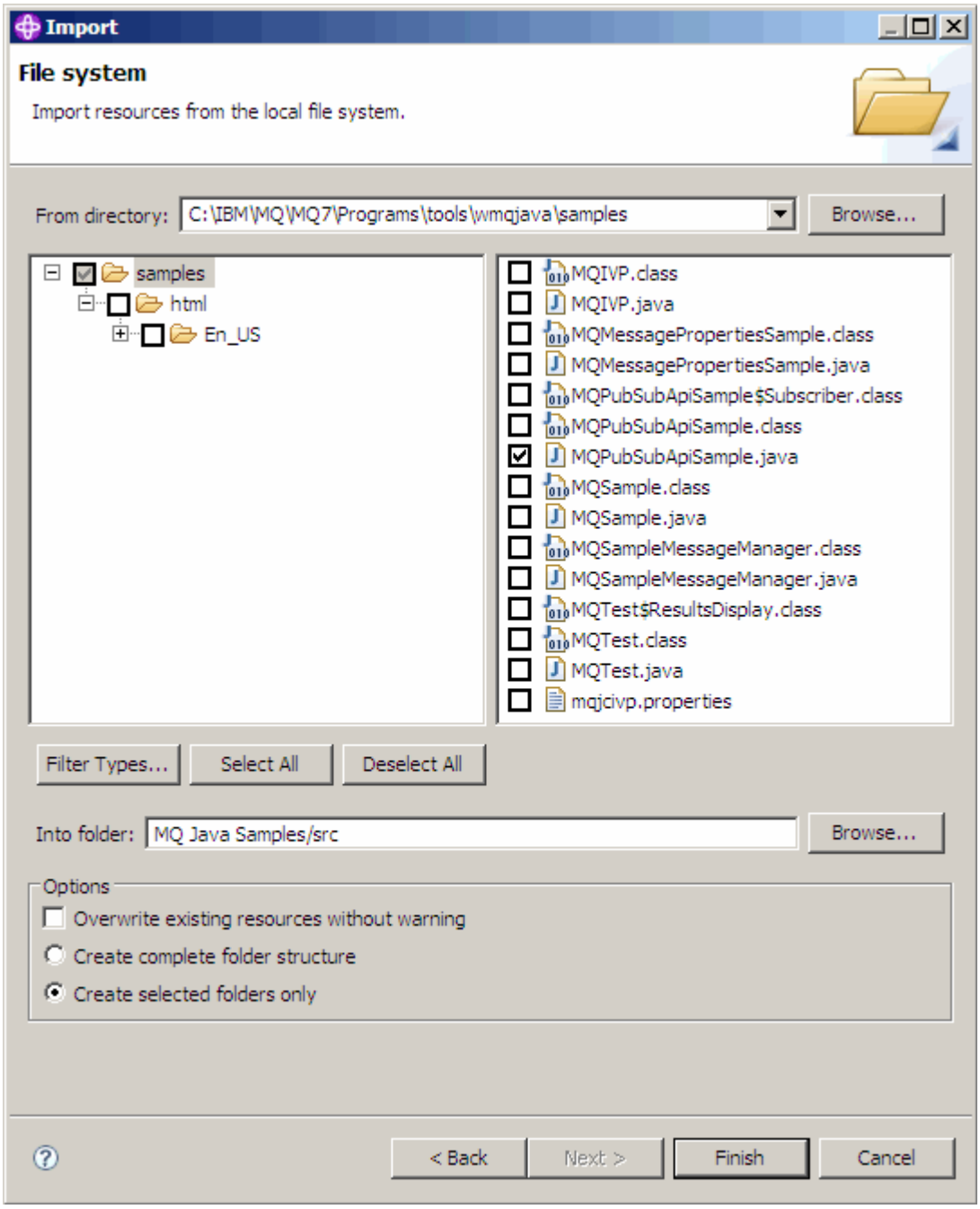


Figura 130. Importación de sistema de archivos

2. Ejecute el programa de ejemplo de publicación/suscripción.

   Hay dos maneras de ejecutar el programa, en función de si necesita cambiar o no los parámetros predeterminados.

   - La primera opción ejecuta el programa sin realizar ningún cambio:
     - En el menú principal del espacio de trabajo, expanda la carpeta `src`. Pulse con el botón derecho del ratón en **MQPubSubApiSample.java Ejecutar como** > **1. Java Aplicación**
   - La segunda opción ejecuta el programa con parámetros o con código fuente modificado para su entorno:
     - Abra `MQPubSubApiSample.java` y estudie el constructor de `MQPubSubApiSample`.

– Modifique los atributos del programa.

  Estos atributos se pueden modificar utilizando el conmutador -D de la JVM, o proporcionando un valor predeterminado para la propiedad System p editando el código fuente.

  - topicObject
  - queueManagerName
  - subscriberCount

  Estos atributos solo son modificables editando el código fuente en el constructor.

  - hostname
  - port
  - canal

  Para establecer las propiedades de System p, codifique un valor predeterminado en el descriptor de acceso, por ejemplo:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  O proporcione el parámetro a la JVM utilizando la opción `-D`, según se muestra en los pasos siguientes:

a. Copie el nombre completo de la propiedad System.Property que desee establecer, por ejemplo: `com.ibm.mq.pubSubSample.queueManagerName`.
b. En el espacio de trabajo, pulse el botón derecho del ratón sobre **Ejecutar** > **Abrir diálogo de ejecución**. Realice una doble pulsación en Aplicación Java en **Crear, gestionar y ejecutar aplicaciones** y pulse el separador **(x) = Argumentos**.
c. En el panel **Argumentos de MV:**, escriba `-D` y pegue el nombre de System.property, `com.ibm.mq.pubSubSample.queueManagerName`, seguido por `=QM3`. Pulse **Aplicar** > **Ejecutar**.
d. Añada más argumentos como una lista separada por comas, o como líneas adicionales en el panel, sin separadores de coma.

   Por ejemplo: `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3`, `-Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

El programa de ejemplo Publish exit (salida de publicación)

AMQSPSE0 es un programa C de ejemplo de una salida para la interceptación de una publicación antes de que se entregue a un suscriptor. A continuación, la salida puede alterar, por ejemplo, las cabeceras de mensaje, la carga útil o el destino, o evitar que el mensaje se publique en un suscriptor.

Para ejecutar el ejemplo, realice las tareas siguientes:

1. Configure el gestor de colas:
   - Linux AIX En los sistemas AIX and Linux, añada una stanza como la siguiente al archivo `qm.ini`:

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     donde el módulo es *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM MQ.
   - Windows En Windows establezca los atributos equivalentes en el registro.
2. Asegúrese de que el módulo es accesible para IBM MQ.
3. Reinicie el gestor de colas para activar la configuración.

4. En el proceso de aplicación que se va a rastrear, indique dónde se deben registrar los archivos de rastreo. Por ejemplo:
   - Linux AIX En sistemas AIX and Linux , asegúrese de que el directorio `/var/mqm/trace` existe y exporte la variable de entorno **MQPSE_TRACE_LOGFILE** :

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows En Windows, asegúrese de que el directorio `C:\temp` existe y establezca la variable de entorno **MQPSE_TRACE_LOGFILE** :

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Programas de transferencia de ejemplo

Los programas de ejemplo Put ponen mensajes en una cola utilizando la llamada MQPUT.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Diseño del programa de transferencia de ejemplo

El programa utiliza la llamada MQOPEN con la opción MQOO_OUTPUT para abrir la cola de destino para transferir mensajes.

Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN. Para que el programa sea sencillo, en esta y en las siguientes llamadas MQI, se utilizarán los valores predeterminados para numerosas opciones.

Para cada línea de entrada, el programa lee el texto en un almacenamiento intermedio y utiliza la llamada MQPUT para crear un mensaje de datagrama que contiene el texto de la línea. El programa continúa hasta llegar al final de la entrada o hasta que falla la llamada MQPUT. Si el programa alcanza el final de la entrada, cierra la cola con la llamada MQCLOSE.

Ejecución de los programas de ejemplo de colocación

Ejecución de los ejemplos amqsput y amqsputc

ALW

El ejemplo **amqqsput** es el programa para transferir mensajes utilizando enlaces locales, y el ejemplo **amqsputc** es el programa para transferir mensajes utilizando enlaces de cliente. Estos programas reciben los siguientes parámetros posicionales:

1. El nombre de la cola de destino (obligatorio).
2. El nombre del gestor de colas (opcional).

   Si no se especifica un gestor de colas, **amqsput** se conecta al gestor de colas predeterminado y **amqsputc** se conecta al gestor de colas identificado por la variable de entorno *MQSERVER* o el archivo de definición de canal de cliente.
3. Opciones de apertura (opcional).

   Si no se especifican las opciones de apertura, el ejemplo utiliza el valor 8208, que es la combinación de estas dos opciones:
   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING
4. Opciones de cierre (opcional).

   Si no se especifican opciones de cierre, el ejemplo utiliza el valor 0, que es MQCO_NONE.

5. El nombre del gestor de colas de destino (opcional).

   Si no se especifica un gestor de colas de destino, el campo `ObjectQMgrName` del MQOD se deja en blanco.

6. El nombre de la cola dinámica (optional).

   Si no se especifica un nombre de cola dinámica, el campo `DynamicQName` del MQOD se deja en blanco.

Utilice las variables de entorno siguientes para proporcionar las credenciales que se utilizan para autenticarse con el gestor de colas:

MQSAMP_USER_ID
Establézcalo en el ID de usuario que se utilizará para la autenticación de conexión, si desea utilizar un ID de usuario y una contraseña para autenticarse con el gestor de colas. El programa solicita la contraseña que acompaña al ID de usuario.

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
Establézcalo en un valor que no esté en blanco si desea proporcionar una señal de autenticación para autenticarse con el gestor de colas. El programa solicita la señal de autenticación. Las señales de autenticación sólo las puede utilizar el ejemplo **amqsputc** que utiliza enlaces de cliente.

Para ejecutar estos programas, especifique uno de los mandatos siguientes:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

donde *micola* es el nombre de la cola en la que se van a colocar los mensajes y *nombregc* es el gestor de colas que posee *micola*.

Ejecución del ejemplo amq0put

ALW

La versión en COBOL no recibe ningún parámetro. Se conecta con el gestor de colas predeterminado y, cuando se ejecuta, solicita el nombre de la cola de destino:

```
Please enter the name of the target queue
```

Recibe entrada de StdIn y añade cada línea de entrada a la cola de destino. Una línea en blanco indica que no hay más datos.

Ejecución del ejemplo AMQSPUT4 en C (IBM i)

IBM i

El programa en C AMQSPUT4, solo disponible para la plataforma IBM i, crea mensajes leyendo datos de un miembro de un archivo de origen.

Hay que especificar el nombre del archivo como un parámetro al iniciar el programa. La estructura del archivo tiene que ser esta:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

En la biblioteca QMQMSAMP archivo AMQSDATA miembro PUT se facilita un ejemplo de entrada para los ejemplos de colocación.

Nota: Recuerde que en los nombres de cola se distingue entre mayúsculas y minúsculas. Todas las colas creadas por programa de ejemplo de creación de archivo AMQSAMP4 tienen nombres en mayúscula.

El programa en C coloca los mensajes en la cola nombrada en la primera línea del archivo; se puede utilizar la cola proporcionada SYSTEM.SAMPLE.LOCAL. El programa coloca el texto de cada una de las siguientes líneas del archivo en mensajes de datagrama separados y se para cuando lee una línea en blanco al final del archivo.

Utilizando el archivo de datos de ejemplo, el comando es:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

Ejecución del ejemplo AMQ0PUT4 en COBOL (IBM i)

IBM i

El programa en COBOL AMQ0PUT4, disponible solo en la plataforma IBM i, crea mensajes aceptando datos del teclado.

Para iniciar el programa, invóquelo pasándole el nombre de la cola de destino como parámetro. El programa guarda la entrada del teclado en un búfer y crea un mensaje de datagrama por cada línea de texto. El programa se para cuando se especifica una línea en blanco en el teclado.

Programas de ejemplo de mensaje de referencia

Los ejemplos de mensajes de referencia permiten que se transfiera un objeto de gran tamaño de un nodo a otro (normalmente, en sistemas distintos) sin que sea necesario almacenar el objeto en colas IBM MQ ni en los nodos de origen ni en los de destino.

Se proporciona un conjunto de programas de ejemplo para ilustrar cómo un mensaje de referencia puede colocarse en una cola, ser recibido por salidas de mensaje y sacado de una cola. Los programas de ejemplo utilizan mensajes de referencia para mover archivos. Si desea mover otros objetos como, por ejemplo, bases de datos, o si desea realizar comprobaciones de seguridad, defina su propia salida basándose en el ejemplo amqsxrm.

La versión del programa de ejemplo de salida de mensaje de referencia dependerá de la plataforma en la que ejecute el canal:

- En todas las plataformas, utilice amqsxrma en el extremo emisor.
- Utilice amqsxrma en el extremo receptor si el destinatario se está ejecutando bajo cualquier plataforma salvo IBM i.
- IBM i Si el destinatario se está ejecutando bajo IBM i, utilice amqsxrm4.

.

IBM i *Notas para los usuarios de IBM i*

Para recibir un mensaje de referencia utilizando la salida de mensaje de ejemplo, especifique un archivo en el sistema de archivos raíz IFS o en cualquier subdirectorio para que se pueda crear un archivo de secuencia continua.

La salida de mensaje de ejemplo en IBM i crea el archivo, convierte los datos a EBCDIC y establece la página de códigos a la página de códigos del sistema. A continuación, puede copiar este archivo en el sistema de archivos QSYS.LIB utilizando el mandato CPYFRMSTMF. Por ejemplo:

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

El mandato CPYFRMSTMF no crea el archivo. Debe crearlo antes de ejecutar este mandato.

Si envía un archivo desde QSYS.LIB, no es necesario realizar ningún cambio en los ejemplos. Para cualquier otro sistema de archivos, asegúrese de que el CCSID especificado en el campo CodedCharSetId de la estructura MQRMH coincide con los datos en masa que está enviando.

Al utilizar el sistema de archivos integrado, cree módulos de programa con el conjunto de opciones SYSIFCOPT(*IFSIO). Si desea mover los archivos de registro de base de datos o de longitud fija, defina su propia salida basándose en el ejemplo AMQSXRM4 proporcionado.

El método recomendado para transferir un archivo de base de datos es convertirlo en una estructura IFS, utilizando el mandato CPYTOSTMF y, a continuación, enviar el mensaje de referencia adjuntando el archivo IFS. Si elige transferir un archivo de base de datos haciendo referencia a él desde dentro de IFS, pero no convertirlo en la estructura IFS, debe especificar el nombre de miembro. Si elije este método, la integridad de los datos no está garantizada.

Multi *Ejecución de los ejemplos de mensajes de referencia*

Utilice este ejemplo para averiguar cómo ejecutar la aplicación de ejemplo Mensaje de referencia AMQSPRM en AIX, Linux, and Windowso AMQSPRMA en IBM i. El ejemplo muestra cómo se pueden poner mensajes de referencia en una cola, recibidos por salidas de mensajes, y tomados de una cola.

Los ejemplos de Mensaje de referencia se ejecutan de la forma siguiente:

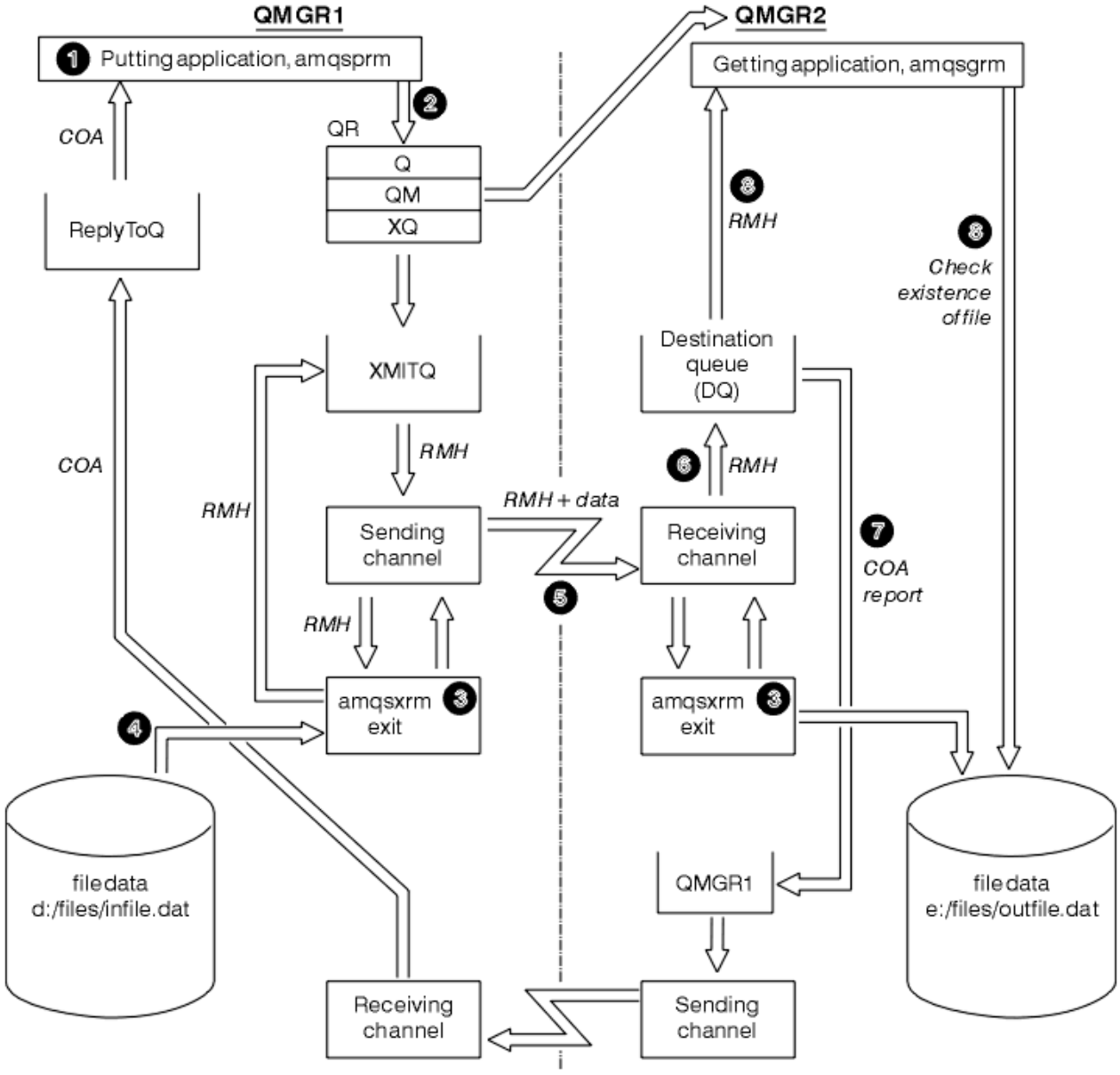


Figura 131. Ejecución de los ejemplos de mensajes de referencia

1. Configure el entorno para iniciar los escuchas, los canales y los supervisores de desencadenante, y defina los canales y las colas.

A los efectos de describir cómo configurar el mensaje de referencia, este ejemplo hace referencia a la máquina de envío como MACHINE1 con un gestor de colas llamado QMGR1 y la máquina receptora como MACHINE2 con un gestor de colas llamado QMGR2.

Nota: Las definiciones siguientes permiten crear un mensaje de referencia para enviar un archivo con un tipo de objeto FLATFILE desde el gestor de colas QMGR1 a QMGR2 y volver a crear el archivo tal como se define en la llamada a AMQSPRM (o AMQSPRMA en IBM i). El mensaje de referencia (incluyendo los datos de archivo) se envía utilizando el canal CHL1 y la cola de transmisión XMITQ y se coloca en la cola DQ. Los informes de excepción y COA se envían de nuevo a QMGR1 utilizando el canal REPORT y la cola de transmisión QMGR1.

La aplicación que recibe el mensaje de referencia (AMQSGRM o AMQSGRMA en IBM i) se desencadena utilizando la cola de inicio INITQ y el proceso PROC. Asegúrese de que los campos CONNAME se hayan establecido correctamente y que el campo MSGEXIT refleje la estructura de directorios, en función del tipo de máquina y del lugar en el que está instalado el producto IBM MQ.

IBM i Las definiciones MQSC han utilizado un estilo AIX para definir las salidas, por lo tanto, si utiliza MQSC en IBM i, debe modificarlas en consecuencia. Es importante darse cuenta de que los datos de mensaje FLATFILE distinguen entre mayúsculas y minúsculas, y el ejemplo no funcionará si no está en mayúsculas.

En la máquina MACHINE1, gestor de colas QMGR1

Sintaxis MQSC

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **Sintaxis del mandato IBM i**

Nota: Si no especifica un nombre de gestor de colas, el sistema utiliza el gestor de colas predeterminado.

```
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)
```

En la máquina MACHINE2, gestor de colas QMGR2

Sintaxis MQSC

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)
```

```
define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i sintaxis del mandato

Nota: On IBM i, si no especifica un nombre de gestor de colas, el sistema utiliza el gestor de colas predeterminado.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. Una vez que se han creado los objetos de IBM MQ:
   a. Cuando corresponda, según la plataforma, inicie la escucha para los gestores de cola de envío y recepción
   b. Inicie los canales CHL1 y REPORT
   c. En el gestor de colas receptor, inicie el supervisor desencadenante para INITQ de la cola de inicio
3. Invoque el programa de ejemplo de referencia de mensajes AMQSPRM (AMQSPRMA en IBM i) desde la línea de mandatos utilizando los parámetros siguientes:

   -m
   Nombre del gestor de colas local; el valor predeterminado es el gestor de colas predeterminado

   -i
   Nombre y ubicación del archivo fuente

   -o
   Nombre y ubicación del archivo de destino

   -q
   Nombre de la cola

   -g
   El nombre del gestor de colas en el que está la cola, definida en el parámetro -q. El valor predeterminado es el gestor de colas especificado en el parámetro -m.

   -t
   Tipo de objeto

   -w
   Intervalo de espera, es decir, el tiempo de espera para los informes de excepción y COA desde el gestor de colas de recepción

   Por ejemplo, para utilizar el ejemplo con los objetos definidos previamente, utilizaría los parámetros siguientes:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

Aumentar el tiempo de espera permite que se envíe un archivo grande a través de una red antes de que el programa ponga los mensajes exceso de tiempo.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

Usuarios de IBM i: IBM i En IBM i, lleve a cabo los pasos siguientes:

a. Utilice el mandato siguiente:

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

   Se presupone que el archivo original `rmsg1` está en el directorio IFS `/refmsgs` y que desea que el archivo de destino sea `rmsgx` en el directorio IFS `/refmsgs` en el sistema de destino.

b. Cree su propio directorio utilizando el mandato CRTDIR en lugar de utilizar el directorio raíz.

c. Cuando llame al programa que pone datos, recuerde que el nombre del archivo de salida debe reflejar el convenio de denominación IFS; por ejemplo, /TEST/FILENAME crea un archivo denominado FILENAME en el directorio TEST.

Nota:

IBM i En IBM i, puede utilizar una barra inclinada (/) o un guión (-) al especificar los parámetros. Por ejemplo:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX Para las plataformas AIX and Linux, debe utilizar dos barras inclinadas invertidas (\\) en lugar de una para indicar el directorio de archivos de destino. Por lo tanto, el mandato **amqsprm** se parece a lo siguiente:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

Al ejecutar el programa de mensajes de referencia put se hace lo siguiente:

- El mensaje de referencia se coloca en la cola QR en el gestor de colas QMGR1.
- El archivo de origen y la vía de acceso son `d:\files\infile.dat` y ya existen en el sistema en el que se emite el mandato de ejemplo.
- Si la cola QR es una cola remota, el mensaje de referencia se envía a otro gestor de cola, en un sistema distinto, en el que se crea un archivo con el nombre y vía de acceso `e:\files\outfile.dat`. El contenido de este archivo es el mismo que el archivo de origen.
- amqsprm espera 30 segundos para un informe de COA desde el gestor de colas de destino.
- El tipo de objeto es `flatfile`, por lo que el canal que se utiliza para mover mensajes de la cola QR debe especificarlo en el campo *MsgData*.

4. Cuando defina los canales, seleccione la salida de mensaje en los extremos de envío y de recepción para que sean amqsxrm.

Windows Esto se define en Windows como se indica a continuación:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Esto se define en AIX and Linux como se indica a continuación:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Si especifica un nombre de vía de acceso, especifique el nombre completo. Si omite el nombre de vía de acceso, se supone que el programa está en la vía de acceso especificada en el archivo `qm.ini` (o, en IBM MQ for Windows, la vía de acceso especificada en el registro).

5. La salida de canal lee la cabecera de mensaje de referencia y encuentra el archivo al que hace referencia.
6. A continuación, la salida de canal puede segmentar el archivo antes de enviarlo por el canal junto con la cabecera.

   Linux AIX En AIX and Linux, cambie el propietario del grupo del directorio de destino a 'mqm' para que la salida de mensaje de ejemplo pueda crear el archivo en ese directorio. Además, cambie los permisos del directorio de destino para permitir que los miembros del grupo mqm escriban en él. Los datos de archivo no se almacenan en las colas de IBM MQ.
7. Cuando el último segmento del archivo se procesa mediante la salida de mensaje de recepción, el mensaje de referencia se coloca en la cola de destino especificada por amqsprm. Si se desencadena esta cola (es decir, la definición especifica los atributos de cola **Trigger**, **InitQ** y **Process**), se desencadena el programa especificado por el parámetro PROC de la cola de destino. El programa a desencadenar debe estar definido en el campo `ApplId` del atributo **Process**.
8. Cuando el mensaje de referencia alcanza la cola de destino (DQ), se envía un informe COA a la aplicación de colocación (amqsprm).
9. El mensaje de referencia Get, amqsgrm, obtiene mensajes de la cola especificada en el mensaje de desencadenante de entrada y comprueba que el archivo existe.

Diseño del ejemplo de colocación de mensaje de referencia (amqsprma.c, AMQSPRM4)
En este tema se proporciona una descripción detallada de un ejemplo de colocación de mensaje de referencia.

En este ejemplo se crea un mensaje de referencia que hace referencia a un archivo y lo coloca en una cola especificada:

1. El ejemplo se conecta con un gestor de colas local utilizando MQCONN.
2. A continuación, abre (MQOPEN) una cola modelo que se utiliza para recibir mensajes de informe.
3. El ejemplo crea un mensaje de referencia que contiene los valores necesarios para mover el archivo, por ejemplo, los nombres de archivo de origen y de destino, y el tipo de objeto. Como muestra, el ejemplo que se entrega con IBM MQ crea un mensaje de referencia para enviar el archivo `d:\x\file.in` desde `QMGR1` a `QMGR2` y para volver a crear el archivo como `d:\y\file.out` utilizando los parámetros siguientes:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   donde `QR` es una definición de cola remota que referencia una cola de destino en `QMGR2`.

   Nota: Para las plataformas AIX and Linux, utilice dos barras inclinadas invertidas (\\) en lugar de una para denotar el directorio del archivo de destino. Por lo tanto, el mandato **amqsprm** se parece a lo siguiente:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```
4. El mensaje de referencia se coloca (sin ningún dato de archivo) en la cola especificada por el parámetro /q. Si se trata de una cola remota, el mensaje se coloca en la cola de transmisión correspondiente.

5. El ejemplo espera, durante el tiempo especificado en el parámetro /w (que toma como valor predeterminado 15 segundos), a que los informes COA, que, junto con los informes de excepción, se devuelven a la cola dinámica creada en el gestor de colas local (QMGR1).

Diseño del ejemplo de una salida del mensaje de referencia (amqsxrma.c, AMQSXRM4)
En este ejemplo se reconocen los mensajes de referencia con un tipo de objeto que coincide con el tipo de objeto en el campo de datos de usuario de salida de mensaje de la definición de canal.

En estos mensajes ocurre lo siguiente:

- En el canal emisor o servidor, la longitud especificada de datos se copia a partir del desplazamiento especificado del archivo especificado en el espacio restante del búfer de agente después del mensaje de referencia. Si no se alcanza el final del archivo, el mensaje de referencia se vuelve a colocar en la cola de transmisión después de actualizar el campo *DataLogicalOffset* .
- En el canal peticionario o receptor, si el campo *DataLogicalOffset* es cero y el archivo especificado no existe, se crea. Los datos que siguen al mensaje de referencia se añaden al final del archivo especificado. Si el mensaje de referencia no es el último del archivo especificado, se descarta. De lo contrario, se devuelve a la salida del canal, sin los datos añadidos, para colocarlo en la cola de destino.

Para los canales emisor y servidor, si el campo *DataLogicalLength* del mensaje de referencia de entrada es cero, la parte restante del archivo, desde *DataLogicalOffset* hasta el final del archivo, se enviará a lo largo del canal. Si no es cero, solo se enviarán la longitud especificada.

Si se produce un error (por ejemplo, si el ejemplo no puede abrir un archivo), MQCXP. *ExitResponse* se establece en MQXCC_SUPPRESS_FUNCTION para que el mensaje que se está procesando se coloque en la cola de mensajes no entregados en lugar de continuar en la cola de destino. Se devuelve un código de comentarios en MQCXP. *Feedback* y se devuelve a la aplicación que ha colocado el mensaje en el campo *Feedback* del descriptor de mensaje de un mensaje de informe. Esto se debe a que la aplicación de transferencia ha solicitado informes de excepción estableciendo MQRO_EXCEPTION en el campo *Report* del MQMD.

Si la codificación *CodedCharacterSetId* (CCSID) del mensaje de referencia es distinta de la del gestor de colas, dicho mensaje se convierte a la codificación y CCSID locales. En este ejemplo, amqsprm, el formato del objeto es MQFMT_STRING, de modo que amqsxrm convierte los datos del objeto al CCSID local del extremo receptor antes de que se escriban en el archivo.

No especifique el formato del archivo que se transfiere como MQFMT_STRING si este contiene caracteres multibyte (por ejemplo, DBCS o Unicode). Esto se debe a que un carácter multibyte podría partirse al segmentarse el archivo en el extremo emisor. Para transferir y convertir un archivo de este tipo, especifique el formato como algo distinto de MQFMT_STRING para que el archivo no se convierta en la salida del mensaje de referencia, sino en el extremo receptor cuando la transferencia se complete.

Compilación del ejemplo Salida de mensaje de referencia
Para compilar el ejemplo de Salida de mensaje de referencia, utilice el mandato correspondiente a la plataforma en la que está instalado IBM MQ.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM MQ.

Para compilar amqsxrma, utilice estos mandatos:

En AIX

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

En IBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

Nota:

1. Para crear el modelo de forma que utilice el sistema de archivos IFS, añada la opción `SYSIFCOPT(*IFSIO)`
2. Para crear el programa para su uso con canales sin hebras, utilice el mandato siguiente: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)`
3. Para crear el programa para su uso con canales con hebras, utilice el mandato siguiente: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)`

En Linux

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

En Windows

Windows IBM MQ proporciona ahora la biblioteca mqm con paquetes de cliente, así como paquetes de servidor, por lo que el ejemplo siguiente utiliza `mqm.lib` en lugar de `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Conceptos relacionados

"Cómo escribir programas de salida de canal" en la página 989
Puede utilizar la información siguiente para ayudarle a escribir programas de salida de canal.

Diseño del ejemplo de obtención de mensaje de referencia (amqsgrma.c, AMQSGRM4)
En este tema se explica el diseño del ejemplo de obtención de mensaje de referencia.

La lógica del programa es la siguiente:

1. El ejemplo se desencadena y extrae los nombres de cola y de gestor de colas del mensaje desencadenante de entrada.
2. Luego se conecta con el gestor de colas especificado usando MQCONN y abre la cola especificada con MQOPEN.
3. El ejemplo emite un MQGET con un intervalo de espera de 15 segundos dentro de un bucle para obtener los mensajes de la cola.
4. Si un mensaje es de referencia, el ejemplo comprueba la existencia del archivo que se ha transferido.
5. Luego cierra la cola y se desconecta del gestor de colas.

Programas de ejemplo de solicitud

Los programas de ejemplo de solicitud ilustran el procesamiento cliente/servidor. Los ejemplos son los clientes que colocan mensajes de solicitud en una cola de servidor de destino procesada por un programa servidor. Esperan a que el programa servidor coloque un mensaje de respuesta en una cola de respuestas.

Los ejemplos de solicitud colocan una serie de mensajes de solicitud en la cola de servidor de destino utilizando la llamada MQPUT. Estos mensajes especifican la cola local (SYSTEM.SAMPLE.REPLY) como cola de respuestas, que puede ser una cola local o remota. Los programas esperan los mensajes de

respuesta y los muestran. Las respuestas solo se envían si una aplicación servidora está procesando la cola del servidor de destino o si se desencadena una aplicación a tal fin (los programas de ejemplo de consulta, establecimiento y eco están diseñados para ser desencadenados). El ejemplo en C espera 1 minuto (el ejemplo en COBOL espera 5 minutos) a que llegue la primera respuesta (para permitir que se desencadene una aplicación de servidor) y 15 segundos para las respuestas posteriores, pero ambos ejemplos pueden terminar sin obtener ninguna respuesta. Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para ver los nombres de los programas de ejemplo de solicitud.

Ejecución de los programas de ejemplo de solicitud

Ejecución de los ejemplos amqsreq0.c, amqsreq y amqsreqc

La versión en C del programa recibe tres parámetros:

1. Nombre de la cola del servidor de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. La cola de respuestas (opcional).

Por ejemplo, especifique una de las opciones siguientes:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

donde `myqueue` es el nombre de la cola del servidor de destino, `qmanagername` es el nombre del gestor de colas propietario de `myqueue` y `replyqueue` es el nombre de la cola de respuestas.

Si se omite el nombre del gestor de colas, se asumirá que el propietario de la cola es el gestor de colas predeterminado. Si se omite el nombre de la cola de respuestas, se proporcionará la cola de respuestas predeterminada.

Ejecución del ejemplo amq0req0.cbl

La versión en COBOL no recibe ningún parámetro. Se conecta con el gestor de colas predeterminado y, cuando se ejecuta, solicita el nombre de la cola de destino:

```
Please enter the name of the target server queue
```

El programa recibe la entrada de StdIn y añade cada línea a la cola del servidor de destino, metiendo cada línea de texto en el contenido de un mensaje de solicitud. El programa finaliza cuando lee una línea nula.

Ejecución del ejemplo AMQSREQ4

El programa en C crea mensajes tomando datos de stdin (el teclado), finalizándose la entrada con una línea en blanco. El programa recibe hasta tres parámetros: el nombre de la cola de destino (obligatorio), el nombre del gestor de colas (opcional) y el nombre de la cola de respuestas (opcional). Si no se especifica ningún nombre de gestor de colas, se utiliza el gestor de colas predeterminado. Si no se especifica ninguna cola de respuestas, se utiliza la cola SYSTEM.SAMPLE.REPLY.

A continuación, se muestra un ejemplo de cómo invocar el programa de ejemplo en C, especificando la cola de respuestas, pero dejando que el gestor de colas sea el predeterminado:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

Nota: Recuerde que en los nombres de cola se distingue entre mayúsculas y minúsculas. Todas las colas creadas por programa de ejemplo de creación de archivo AMQSAMP4 tienen nombres en mayúscula.

Ejecución del ejemplo AMQ0REQ4

El programa en COBOL crea mensajes recibiendo datos del teclado. Para iniciar el programa, invóquelo especificando el nombre de la cola de destino como parámetro. El programa guarda la entrada del teclado en un búfer y crea un mensaje de solicitud por cada línea de texto. El programa se para cuando se especifica una línea en blanco en el teclado.

Ejecutar el ejemplo de solicitud mediante desencadenamiento

Si el ejemplo se utiliza con desencadenamiento y uno de los programas de ejemplo de consulta, definición o repetición, la línea de entrada debe ser el nombre de la cola a la que desea que acceda el programa desencadenado.

ALW *Ejecución del ejemplo de solicitud utilizando el desencadenante en AIX, Linux, and Windows*

En AIX, Linux, and Windows, inicie el programa de supervisor desencadenante RUNMQTRM en una sesión y, después, inicie el programa amqsreq en otra sesión.

Para ejecutar los ejemplos utilizando el desencadenamiento:

1. Inicie el programa supervisor desencadenante RUNMQTRM en una sesión (la cola de inicio SYSTEM.SAMPLE.TRIGGER está disponible para su uso).
2. Inicie el programa amqsreq en otra sesión.
3. Asegúrese de haber definido una cola de servidor de destino.

   Las colas de ejemplo disponibles para su uso como cola de servidor de destino del ejemplo de solicitud para colocar mensajes son:

   - SYSTEM.SAMPLE.INQ para el programa de ejemplo de consulta.
   - SYSTEM.SAMPLE.SET para el programa de ejemplo de configuración.
   - SYSTEM.SAMPLE.ECHO para el programa de ejemplo de eco.

   Estas colas de ejemplo tienen un tipo de desencadenante FIRST, por lo que si hay mensajes en las colas antes de ejecutar el ejemplo de solicitud, los mensajes enviados no desencadenarán las aplicaciones de servidor.
4. Asegúrese de haber definido una cola para su uso en los programas de ejemplo de consulta, configuración y eco.

Esto significa que el supervisor desencadenante estará preparado cuando el ejemplo de solicitud envíe un mensaje.

Nota: Las definiciones de proceso de ejemplo creadas con RUNMQSC y el archivo amqscos0.tst desencadenan los ejemplos en C. Cambie las definiciones de proceso en amqscos0.tst y utilice RUNMQSC con este archivo actualizado para utilizar las versiones en COBOL.

Figura 132 en la página 1144 ilustra cómo utilizar los ejemplos de solicitud y consulta juntos.

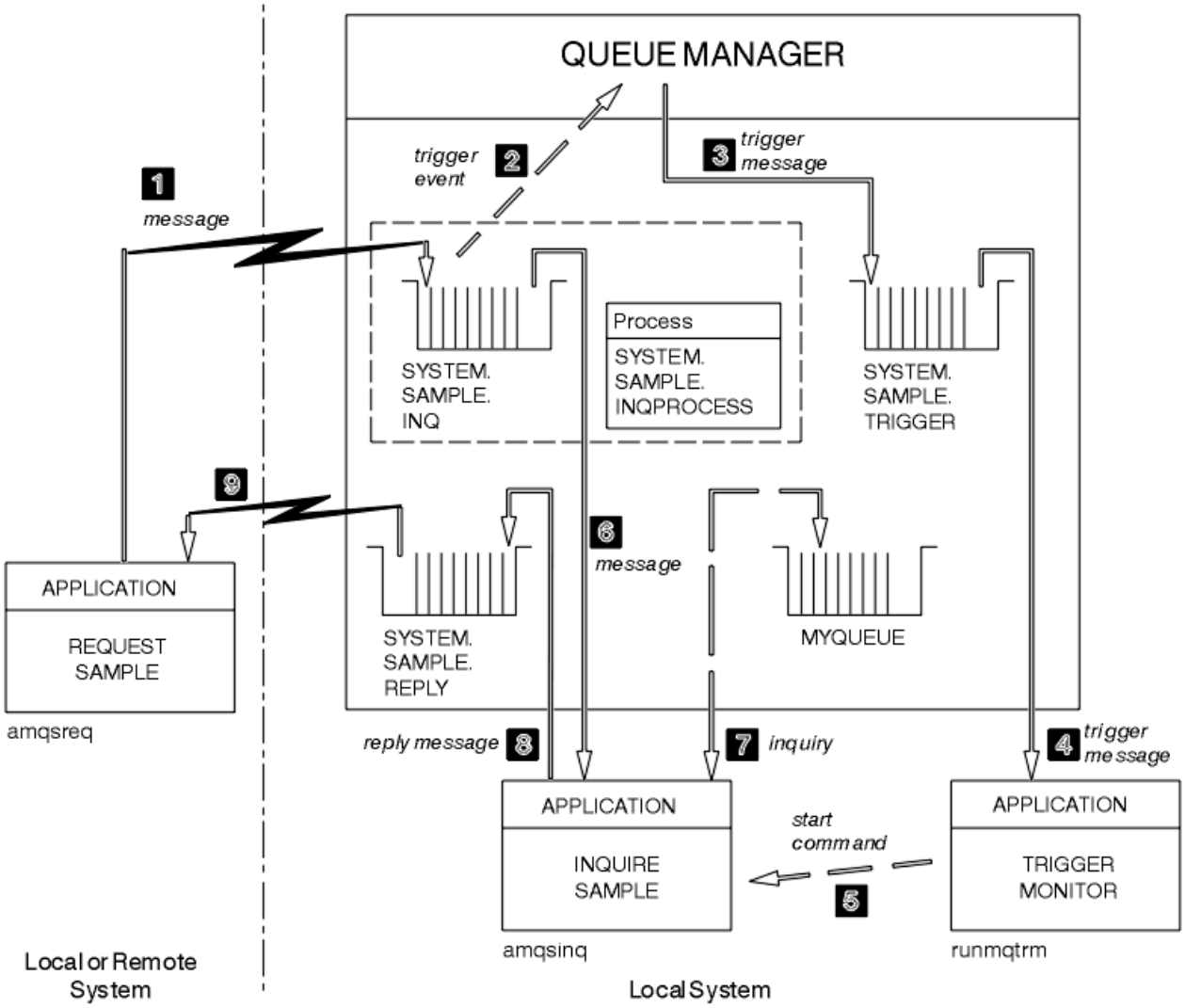


Figura 132. Ejemplos de solicitud y consulta que utilizan el desencadenamiento

En Figura 132 en la página 1144, el ejemplo de solicitud coloca mensajes en la cola del servidor de destino, SYSTEM.SAMPLE.INQ, y el ejemplo de consulte consulta la cola, MYQUEUE. De forma alternativa, se puede utilizar una de las colas de ejemplo definidas al ejecutar amqscos0.tst, o cualquier otra cola que se haya definido, en el ejemplo de consulta.

Nota: Los números de Figura 132 en la página 1144 muestran la secuencia de sucesos.

Para ejecutar los ejemplos de solicitud y consulta con desencadenamiento:

1. Compruebe que están definidas las colas que desee utilizar. Ejecute amqscos0.tst para definir las colas de ejemplo y defina la cola MYQUEUE.
2. Ejecute el comando del supervisor desencadenante RUNMQTRM:

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. Ejecute el ejemplo de solicitud.

```
amqsreq SYSTEM.SAMPLE.INQ
```

Nota: El objeto de proceso define lo que tiene que desencadenarse. Si el cliente y el servidor no se ejecutan en la misma plataforma, los procesos iniciados por el supervisor desencadenante deben

definir *ApplType*; de lo contrario, el servidor toma sus definiciones predeterminadas (es decir, el tipo de aplicación que normalmente está asociada con la máquina del servidor) y provoca una anomalía.

Para obtener una lista de los tipos de aplicación, consulte ApplType.

4. Especifique el nombre de la cola que desee que se utilice en el ejemplo de consulta:

```
MYQUEUE
```

5. Especifique una línea en blanco (para terminar el programa de solicitud).
6. A continuación, el ejemplo de solicitud mostrará un mensaje que contiene los datos que el programa de consulta ha obtenido de MYQUEUE.

Puede utilizar más de una cola; en este caso, entre los nombres de las otras colas en el paso "4" en la página 1145.

Para obtener más información sobre el desencadenante, consulte "Inicio de aplicaciones de IBM MQ utilizando desencadenantes" en la página 886.

IBM i *Ejecución del ejemplo de solicitud utilizando el desencadenante en IBM i*

En IBM i, inicie el servidor desencadenante de ejemplo, AMQSERV4, en un trabajo y, a continuación, inicie AMQSREQ4 en otro. Esto significa que el servidor desencadenante estará preparado cuando el programa de ejemplo de solicitud envía un mensaje.

Nota:

1. Las definiciones de ejemplo creadas por AMQSAMP4 desencadenan las versiones C de los ejemplos. Si quiere desencadenar las versiones COBOL, cambie las definiciones de procesos SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESS y SYSTEM.SAMPLE.SETPROCESS. Puede utilizar el mandato CHGMQMPRC (para obtener detalles, consulte Cambiar proceso MQ (CHGMQMPRC)) para hacer esto, o edite y ejecute su propia versión de AMQSAMP4.
2. El código fuente de AMQSERV4 se proporciona sólo para el lenguaje C. Sin embargo, se proporciona una versión compilada (que se puede utilizar con los ejemplos COBOL) en la biblioteca QMQM.

Puede poner los mensajes de solicitud en estas colas de servidor de ejemplo:

- SYSTEM.SAMPLE.ECHO (para los programas de ejemplo Echo)
- SYSTEM.SAMPLE.INQ (para los programas de ejemplo Inquire)
- SYSTEM.SAMPLE.SET (para los programas de ejemplo Set).

Verá un diagrama de flujo del programa SYSTEM.SAMPLE.ECHO en la Figura 133 en la página 1147. Al utilizar el archivo de datos de ejemplo, el mandato para emitir la solicitud de programa C a este servidor es:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Nota: Esta cola de ejemplo tiene el tipo de desencadenante FIRST, por lo que si hay mensajes en las colas antes de ejecutar el ejemplo Request, los mensajes enviados no desencadenarán las aplicaciones de servidor.

Si desea intentar otros ejemplos, puede probar las siguientes variaciones:

- Utilice AMQSTRG4 (o su línea de mandatos equivalente STRMQMTRM; para obtener detalles, consulte Iniciar supervisor de desencadenante de MQ (STRMQMTRM)) en lugar de AMQSERV4 para enviar el trabajo en su lugar, pero los posibles retrasos en el envío de trabajos podrían hacer que no sea fácil seguir lo que está sucediendo.
- Ejecute los programas de ejemplo SYSTEM.SAMPLE.INQUIRE y SYSTEM.SAMPLE.SET. Con el archivo de datos de ejemplo, los mandatos para emitir las solicitudes de programa C a estos servidores son, respectivamente:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Estas colas de ejemplo también tienen el tipo de desencadenante FIRST.

Diseño del programa de ejemplo de solicitud
El programa abre la cola de servidor de destino para que pueda colocar mensajes. Utiliza la llamada MQOPEN con la opción MQOO_OUTPUT. Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN.

A continuación, el programa abre la cola de respuestas llamada SYSTEM.SAMPLE.REPLY para que se puedan obtener los mensajes de respuesta. Para ello, el programa utiliza la llamada MQOPEN con la opción MQOO_INPUT_EXCLUSIVE. Si no puede abrir la cola, el programa muestra un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN.

Por cada línea de entrada, el programa lee el texto en un búfer y utiliza la llamada MQPUT para crear un mensaje de solicitud que contiene el texto de dicha línea. En esta llamada, el programa utiliza la opción de informe MQRO_EXCEPTION_WITH_DATA para solicitar que cualquier mensaje de informe enviado relativo al mensaje de petición incluya los primeros 100 bytes de los datos de mensaje. El programa continúa hasta llegar al final de la entrada o hasta que falla la llamada MQPUT.

A continuación, el programa utiliza la llamada MQGET para eliminar los mensajes de respuesta de la cola y visualiza los datos contenidos en las respuestas. La llamada MQGET usa las opciones MQGMO_WAIT, MQGMO_CONVERT y MQGMO_ACCEPT_TRUNCATED. *WaitInterval* (intervalo de espera) es de 5 minutos en la versión de COBOL y 1 de minuto en la versión C en la primera respuesta (para dar tiempo a que se desencadene una aplicación de servidor) y de 15 segundos en las respuestas posteriores. El programa espera estos periodos si no hay ningún mensaje en la cola. Si no llega ningún mensaje antes de que venza este intervalo, la llamada falla y devuelve el código de razón MQRC_NO_MSG_AVAILABLE. La llamada también utiliza la opción MQGMO_ACCEPT_TRUNCATED_MSG, de modo que los mensajes más largos que el búfer declarado se truncan

El programa muestra cómo borrar los campos *MsgId* y *CorrelId* de la estructura MQMD después de cada llamada MQGET porque la llamada establece estos campos en los valores contenidos en el mensaje que recupera. Si se limpian estos campos, sucesivas llamadas MQGET recuperarán los mensajes en el orden en que estén guardados en la cola.

El programa continúa hasta que la llamada MQGET devuelve el código de razón MQRC_NO_MSG_AVAILABLE o falla. Si la llamada falla, el programa muestra un mensaje de error que contiene el código de razón.

A continuación, el programa cierra tanto la cola de servidor de destino como la cola de respuestas con la llamada MQCLOSE.

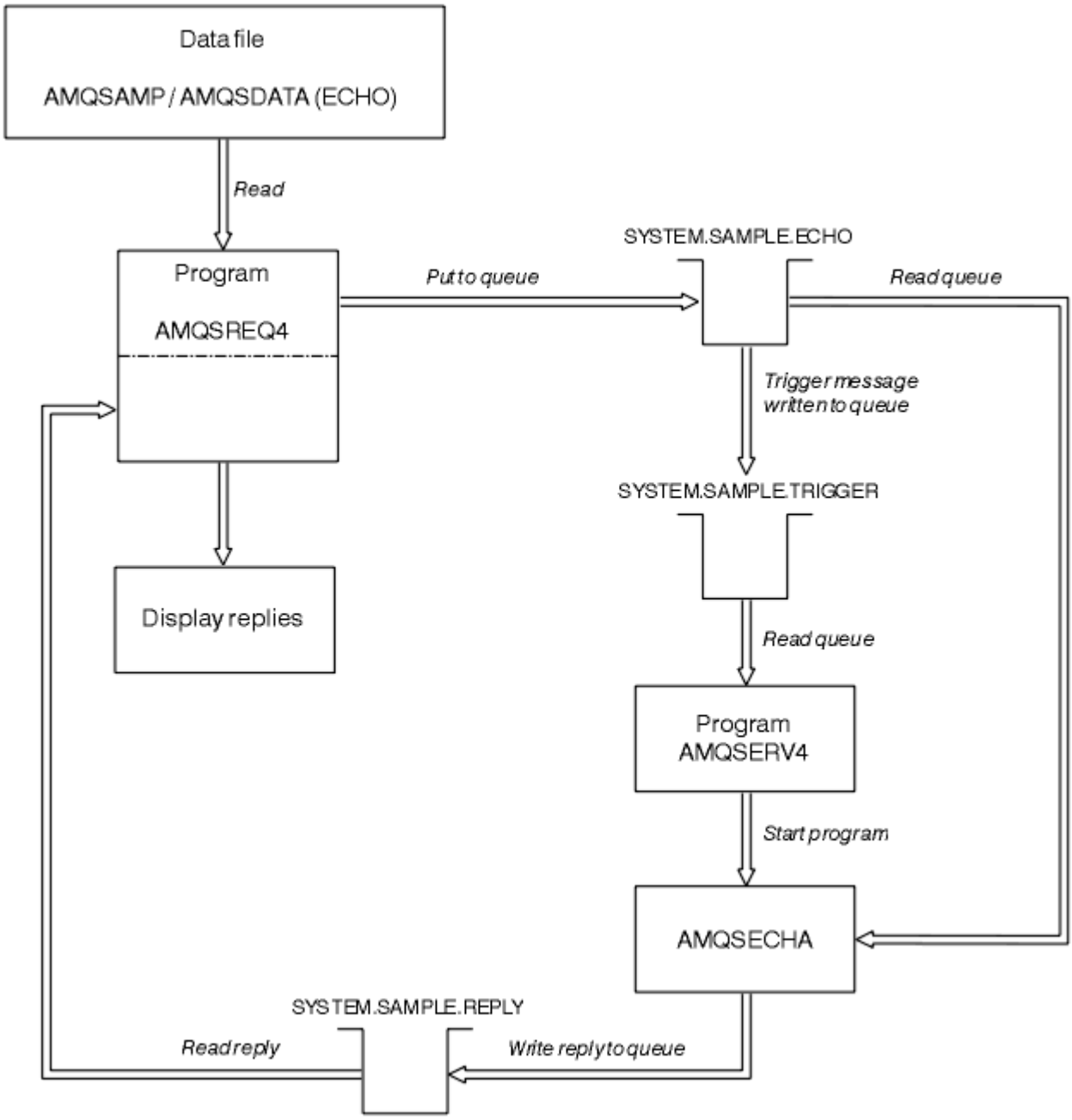


Figura 133. Diagrama de flujo del programa de ejemplo de cliente/servidor IBM i (eco)

Programas Set de ejemplo

Los programas Set de ejemplo inhiben las operaciones de colocación en una cola utilizando la llamada MQSET para cambiar el atributo **InhibitPut** de la cola. Obtenga también información acerca del diseño de los programas Set de ejemplo.

Consulte "Funciones que se ilustran en los programas de ejemplo en Multiplatforms" en la página 1087 para los nombres de estos programas.

Los programas están diseñados para ejecutarse como programas desencadenados, de modo que su única entrada es una estructura MQTMC2 (mensaje desencadenante) que contiene el nombre de una cola de destino con atributos que se han de consultar. La versión en C también utiliza el nombre del gestor de colas. La versión en COBOL utiliza el gestor de colas predeterminado.

Para que funcione el proceso desencadenante, asegúrese de que el programa de ejemplo Set que desea utilizar lo desencadenan los mensajes que llegan a la cola SYSTEM.SAMPLE.SET. Para ello, especifique

el nombre del programa de ejemplo Set que desea utilizar en el campo *ApplicId* de la definición de proceso SYSTEM.SAMPLE.SETPROCESS. La cola de ejemplo tiene el tipo de desencadenante FIRST, por lo que si hay mensajes en la cola antes de que se ejecute el ejemplo Request, los mensajes que envíe no desencadenarán el ejemplo Set.

Cuando haya configurado correctamente la definición:

- ALW En el caso de los sistemas AIX, Linux, and Windows, inicie el programa **runmqtrm** en una sesión, a continuación, inicie el programa amqsreq en otra sesión.
- IBM i En IBM i, inicie el programa AMQSERV4 en una sesión y, a continuación, inicie el programa AMQSREQ4 en otra sesión. Puede utilizar AMQSTRG4, en lugar de AMQSERV4 pero, debido a que pueden producirse retardos en el envío de trabajos, es posible que no le resulte tan fácil seguir lo que está sucediendo.

Utilice los programas de ejemplo Request para enviar mensajes de solicitud a la cola SYSTEM.SAMPLE.SET, cada uno de los mensajes simplemente con un nombre de cola. Para cada mensaje de solicitud, los programas de ejemplo envían un mensaje de respuesta que contiene una confirmación de que se han inhibido las operaciones de colocación en cola para la cola especificada. Las respuestas se envían a la cola de respuestas especificada en el mensaje de solicitud.

Diseño del programa de ejemplo Set

El programa abre la cola referenciada en la estructura de mensajes de desencadenante que se le pasó al iniciarlo. (Para que quede más claro, esta cola se llamará *cola de solicitudes*). El programa usa la llamada MQOPEN para abrir esta cola para entrada compartida.

El programa utiliza la llamada MQGET para eliminar mensajes de esta cola. En esta llamada se utilizan las opciones MQGMO_ACCEPT_TRUNCATED_MSG y MQGMO_WAIT con un intervalo de espera de 5 segundos. El programa prueba el descriptor de cada mensaje para saber si es un mensaje de solicitud. Si no lo es, descarta el mensaje y visualiza un mensaje de aviso.

En cada mensaje de solicitud que se ha eliminado de la cola de solicitudes, el programa lee el nombre de la cola, que hemos denominado la *cola de destino*, que figura en los datos y abre dicha cola utilizando la llamada MQOPEN con la opción MQOO_SET. A continuación, el programa utiliza la llamada MQSET para establecer el valor del atributo **InhibitPut** de la cola de destino en MQQA_PUT_INHIBITED.

Si la llamada MQSET se ejecuta correctamente, el programa utiliza MQPUT1 para colocar un mensaje de respuesta en la cola de respuesta. Este mensaje contiene la serie `PUT inhibited`.

Si la llamada MQOPEN o MQSET no se ejecuta correctamente, el programa utiliza la llamada MQPUT1 para colocar el mensaje `report` en la cola de respuesta. En el campo *Feedback* del descriptor de mensaje de este mensaje de informe se encuentra el código de razón devuelto por la llamada MQOPEN o MQSET, en función de cuál haya fallado.

Después de la llamada MQSET, el programa cierra la cola de destino utilizando la llamada MQCLOSE.

Cuando no quedan mensajes en la cola de solicitudes, el programa cierra esa cola y se desconecta del gestor de colas.

El programa de ejemplo TLS

AMQSSSLC es un programa C de ejemplo que muestra cómo utilizar las estructuras MQCNO y MQSCO para proporcionar información de conexiones de cliente TLS en la llamada MQCONNX. Esto permite que una aplicación MQI de cliente proporcione la definición de su canal de conexiones de cliente y los valores TLS durante la ejecución sin una tabla de definición de canales de cliente (CCDT).

Si se proporciona un nombre de conexión, el programa crea una definición de canal de conexión de cliente en una estructura MQCD.

Si se proporciona el nombre de la raíz del archivo del repositorio de claves, el programa crea una estructura MQSCO. Si también se proporciona un URL de respuesta OCSP, el programa crea una estructura MQAIR de registro de información de autenticación.

A continuación, el programa se conecta al gestor de colas utilizando MQCONNX. Consulta y muestra el nombre del gestor de colas al que se ha conectado.

Este programa se ha diseñado para que se enlace como una aplicación de cliente MQI. Sin embargo, se puede enlazar como una aplicación MQI normal, en cuyo caso simplemente se conecta a un gestor de colas local e ignora la información de conexión de cliente.

V 9.3.0 V 9.3.0 Si la frase de contraseña para acceder al repositorio de claves no está oculta en un archivo, debe proporcionar la frase de contraseña a **amqssslc** cuando se ejecute la aplicación. Puede proporcionar la frase de contraseña:

- Solicitando a **amqssslc** que solicite la frase de contraseña, o
- Utilizando la variable de entorno *MQKEYRPWD* , o
- Utilización del atributo **SSLKeyRepositoryPassword** en el archivo de configuración del cliente

Para obtener más información sobre cómo proporcionar la contraseña del repositorio de claves a las aplicaciones IBM MQ MQI client , consulte Suministro de la contraseña del repositorio de claves para un IBM MQ MQI client en AIX, Linux, and Windows.

amqssslc acepta los parámetros siguientes, todos los cuales son opcionales:

-m QmgrName
El nombre del gestor de colas al que se va a conectar

-c ChannelName
El nombre del canal que se va a utilizar

-x ConnName
El nombre de conexión del servidor

Parámetros TLS:

V 9.3.0 V 9.3.0 **-k KeyReposFileName**
El nombre del archivo de repositorio de claves. Si no se proporciona la extensión de archivo, se presupone que es .kdb. Por ejemplo:

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
La serie CipherSpec del canal TLS correspondiente a **SSLCIPH** en la definición de canal SVRCONN en el gestor de colas.

-f
Especifica que solo se deben utilizar algoritmos FIPS 140-2 certificados.

-b VALUE1[,VALUE2...]
Especifica que solo se deben utilizar algoritmos compatibles con Suite B. Este parámetro es una lista separada por comas de uno o varios valores: NONE,128_BIT,192_BIT. Estos valores tienen el mismo significado que los de la variable de entorno **MQSUITEB** y el valor **EncryptionPolicySuiteB** equivalente en la stanza SSL del archivo de configuración del cliente.

-p Policy
Especifica la política de validación de certificados que se ha de utilizar. Puede tener uno de los valores siguientes:

CUALQUIERA
Aplicar cada política de validación de certificados soportada por la biblioteca de sockets seguros y aceptar la cadena de certificados si cualquiera de las políticas considera válida la cadena de certificados. Este valor se puede utilizar para lograr la máxima compatibilidad con certificados digitales más antiguos que no cumplen las normas modernas para certificados.

RFC5280
Esta opción aplica sólo la política de validación de certificados compatible con RFC 5280. Este valor proporciona una validación más estricta que el valor ANY, pero rechaza algunos certificados digitales más antiguos.

El valor predeterminado es ANY.

-l CertLabel
La etiqueta de certificado que se debe utilizar para la conexión segura.

Nota: Debe especificar el valor utilizando caracteres en minúsculas.

V 9.3.0 V 9.3.0 **-w**
Especifica que **amqssslc** solicita que se proporcione la frase de contraseña del repositorio de claves.

V 9.3.0 V 9.3.0 **-i**
Especifica que **amqssslc** solicita la clave inicial utilizada para cifrar la frase de contraseña del repositorio de claves que se va a proporcionar.

Especifique esta opción si se ha especificado un archivo de claves inicial cuando se cifró la frase de contraseña del repositorio de claves utilizando el programa de utilidad **runmqicred** .

Parámetro de revocación de certificados OCSP:

-o URL
El URL de respondedor OCSP

También puede establecer una de las variables de entorno siguientes para proporcionar credenciales que se utilizan para autenticarse con el gestor de colas:

MQSAMP_USER_ID
Establézcalo en el ID de usuario que se utilizará para la autenticación de conexión, si desea utilizar un ID de usuario y una contraseña para autenticarse con el gestor de colas. El programa solicita la contraseña que acompaña al ID de usuario.

Linux AIX V 9.3.4 **MQSAMP_TOKEN**
Establézcalo en un valor que no esté en blanco si desea proporcionar una señal de autenticación para autenticarse con el gestor de colas. El programa solicita la señal de autenticación.

Ejecución del programa de ejemplo TLS
Para ejecutar el programa de ejemplo TLS, primero hay que configurar el entorno TLS. Luego ejecute el ejemplo por línea de comandos, pasándole varios parámetros.

Acerca de esta tarea

En las instrucciones siguientes se ejecuta el programa de ejemplo utilizando certificados personales. Cambiando el comando se puede, por ejemplo, utilizar certificados de CA y comprobar su estado con un respondedor OCSP. Consulte las instrucciones que se incluyen en el ejemplo.

Procedimiento

1. Cree un gestor de colas de nombre QM1. Si desea más información, consulte crtmqm.
2. Cree un repositorio de claves para el gestor de colas. Para obtener más información, consulte Configuración de un repositorio de claves en AIX, Linux, and Windows.
3. Cree un repositorio de claves para el cliente. Llámelo *clientkey.kdb*.

   Oculte la contraseña del repositorio de claves en un archivo al crear el repositorio de claves.
4. Cree un certificado personal para el gestor de colas. Para obtener más información, consulte Creación de un certificado personal autofirmado en AIX, Linux, and Windows.
5. Cree un certificado personal para el cliente.
6. Extraiga el certificado personal del repositorio de claves del servidor y añádalo al repositorio cliente. Para obtener más información, consulte Extracción de la parte pública de un certificado autofirmado de un repositorio de claves en AIX, Linux, and Windowsy Adición de un certificado de CA (o la

parte pública de un certificado autofirmado) en un repositorio de claves, en sistemas AIX, Linux, and Windows.

7. Extraiga el certificado personal del repositorio de claves del cliente y añádalo al repositorio de claves del servidor.
8. Cree un canal de conexión de servidor con el comando MQSC:

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

   Para obtener más información, consulte Canal de conexión del servidor
9. Defina e inicie un escucha de canal en el gestor de colas. Puede obtener información adicional consultando DEFINE LISTENER y START LISTENER.
10. Ejecute el programa de ejemplo con el comando siguiente:

V 9.3.0 V 9.3.0

```
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:\Archivos de programa\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

Resultados

El programa de ejemplo realiza las acciones siguientes:

1. Se conecta con cualquier gestor de colas que se especifique o con el gestor de colas predeterminado, utilizando las opciones que se especifiquen.
2. Abre el gestor de colas y consulta su nombre.
3. Cierra el gestor de colas.
4. Desconecta del gestor de colas.

Si el programa de ejemplo ejecuta correctamente, mostrará una la salida similar a la del ejemplo siguiente:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Archivos de programa\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Si el programa de ejemplo encuentra un problema, muestra el correspondiente mensaje de error; por ejemplo, si se especifica un URL de respondedor OCSP no válido, se recibirá el mensaje siguiente:

```
MQCONNX ended with reason code 2553
```

Para obtener una lista de códigos de razón, consulte Códigos de terminación y razón de la API.

Programas de ejemplo de desencadenamiento

La función que se proporciona en el ejemplo de desencadenamiento es un subconjunto de la que se proporciona en el supervisor desencadenante en el programa **runmqtrm**.

Consulte “Funciones que se ilustran en los programas de ejemplo en Multiplatforms” en la página 1087 para los nombres de estos programas.

Diseño del ejemplo de desencadenamiento

El programa de ejemplo de desencadenamiento abre la cola de inicio usando la llamada MQOPEN con la opción MQOO_INPUT_AS_Q_DEF. Obtiene mensajes de la cola de inicio utilizando la llamada MQGET con las opciones MQGMO_ACCEPT_TRUNCATED_MSG y MQGMO_WAIT, especificando un intervalo de espera ilimitado. El programa borra los campos *MsgId* y *CorrelId* antes de cada llamada MQGET para obtener mensajes en secuencia.

Cuando ha recuperado un mensaje de la cola de inicio, el programa verifica el mensaje comprobando que su tamaño es el de una estructura MQTM. Si dicha comprobación falla, el programa muestra una advertencia.

En los mensajes desencadenantes válidos, el ejemplo de desencadenamiento copia los datos de estos campos: *ApplicId*, *EnvrData*, *Version* y *ApplType*. Los dos últimos de estos campos son numéricos, así que el programa crea sustituciones de caracteres para utilizar en una estructura MQTMC2 para los sistemas IBM i, AIX, Linux, and Windows.

El ejemplo de desencadenamiento emite un comando de inicio a las aplicaciones especificadas en el campo *ApplicId* del mensaje desencadenante y pasa una estructura MQTMC2 o MQTMC (una versión en caracteres del mensaje desencadenante).

- ALW En los sistemas AIX, Linux, and Windows, el campo *EnvData* se utiliza como una extensión de la serie de mandatos de invocación.
- IBM i En IBM i, se utiliza como parámetros de envío de trabajo, por ejemplo, la prioridad del trabajo o la descripción del trabajo.

Por último, el programa cierra la cola de inicio.

Terminación de los programas de ejemplo de desencadenante en IBM i

IBM i

Un programa de supervisor desencadenante puede ser finalizado por la opción 2 de sysrequest (ENDRQS) o inhibiendo las obtenciones de la cola de desencadenantes.

Si se utiliza la cola de desencadenantes del ejemplo, el comando es:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

Importante: Antes de reiniciar el desencadenamiento en esta cola, hay que ejecutar el comando siguiente:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Ejecución de los programas de ejemplo de desencadenamiento
Este tema contiene información sobre la ejecución de programas de ejemplo de desencadenamiento.

Ejecución de los ejemplos amqstrg0.c, amqstrg y amqstrgc

El programa recibe 2 parámetros:

1. El nombre de la cola de inicio (obligatorio).
2. El nombre del gestor de colas (opcional).

Si no se especifica ningún gestor de colas, se conecta con el predeterminado. Se habrá definido una cola de inicio de ejemplo al ejecutar amqscos0.tst; el nombre de dicha cola es SYSTEM.SAMPLE.TRIGGER y puede usarse al ejecutar este programa.

Nota: La función de este ejemplo es un subconjunto de la función de desencadenamiento completa que se proporciona en el programa runmqtrm.

Ejecución del ejemplo AMQSTRG4

IBM i

Se trata de un supervisor desencadenante para el entorno IBM i. Envía un trabajo IBM i para cada aplicación que se va a iniciar. Esto significa que hay un proceso adicional asociado a cada mensaje desencadenante.

AMQSTRG4 (en QCSRC) recibe dos parámetros: el nombre de la cola de inicio a la que va a dar servicio y el nombre del gestor de colas (opcional). AMQSAMP4 (en QCLSRC) define un ejemplo de cola de inicio, SYSTEM.SAMPLE.TRIGGER, que se podrá utilizar al intentar ejecutar los programas de ejemplo.

Al utilizar la cola desencadenante de ejemplo, el comando que se ha de emitir es:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

De forma alternativa, se puede utilizar el equivalente de CL STRMQMTRM; para obtener detalles, consulte Inicio del supervisor desencadenante de MQ (STRMQMTRM).

Ejecución del ejemplo AMQSERV4

IBM i

Se trata de un servidor desencadenante para el entorno IBM i. Por cada mensaje desencadenante, este servidor ejecuta el comando de inicio en su propio trabajo para iniciar la aplicación especificada. El servidor desencadenante puede invocar las transacciones CICS.

AMQSERV4 recibe dos parámetros: el nombre de la cola de inicio a la que va a dar servicio y el nombre del gestor de colas (opcional). AMQSAMP4 define un ejemplo de cola de inicio, SYSTEM.SAMPLE.TRIGGER, que se podrá utilizar al intentar ejecutar los programas de ejemplo.

Al utilizar la cola desencadenante de ejemplo, el comando que se ha de emitir es:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Diseño del servidor desencadenante
El diseño del servidor desencadenante es similar al del supervisor desencadenante, con algunas excepciones

El diseño del servidor desencadenante es similar al del supervisor desencadenante, salvo que el servidor desencadenante:

- Permite aplicaciones MQAT_CICS y MQAT_OS400.
- IBM i Invoca aplicaciones de IBM i en su propio trabajo (o utiliza STRCICSUSR para iniciar aplicaciones de CICS) en lugar de enviar un trabajo IBM i.
- En el caso de las aplicaciones CICS, sustituye *EnvData*, por ejemplo, para especificar la región CICS, del mensaje desencadenante en el comando STRCICSUSR.
- Abre la cola de inicio de la entrada compartida, de modo que muchos servidores desencadenantes pueden ejecutar a la vez.

Nota: Los programas iniciados por AMQSERV4 no pueden utilizar la llamada MQDISC, porque esto para el servidor desencadenante. Si los programas iniciados por AMQSERV4 utilizan la llamada MQCONN, obtienen el código de razón MQRC_ALREADY_CONNECTED.

ALW *Utilización de los ejemplos de TUXEDO en AIX, Linux, and Windows*

Obtenga información sobre los programas de ejemplo Put y Get de TUXEDO y sobre cómo crear el entorno de servidor en TUXEDO.

Antes de empezar

Antes de ejecutar estos ejemplos, debe crear el entorno de servidor.

Acerca de esta tarea

Nota: A lo largo de esta sección, se utiliza el carácter de barra inclinada invertida (\) para dividir los mandatos largos en más de una línea. No especifique este carácter. Entre cada mandato como una sola línea.

ALW *Creación del entorno de servidor*
Información sobre cómo crear el entorno de servidor para IBM MQ para distintas plataformas.

Antes de empezar

Se asume que existe un entorno TUXEDO operativo.

AIX *Compilación del entorno de servidor en AIX (32 bits)*
Cómo compilar el entorno de servidor en IBM MQ for AIX (32 bits).

Procedimiento

1. Cree un directorio (por ejemplo, APPDIR) en el que se compile el entorno de servidor y ejecute todos los comandos en ese directorio.
2. Exporte las variables de entorno siguientes, donde TUXDIR es el directorio raíz para TUXEDO, y *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el cual está instalado IBM MQ:

```
$ 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. Añada la línea siguiente al archivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Ejecute los comandos:

```
$ 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. Edite ubbstxcx.cfg y añada los detalles del nombre de máquina, los directorios de trabajo y el gestor de colas según sea necesario:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Cree el TLOGDEVICE:

```
$tmadmin -c
```

Aparece un símbolo del sistema. En dicho símbolo, especifique:

```
> crdl -z /APPDIR/TLOG1
```

7. Inicie el gestor de colas:

```
$ strmqm
```

8. Inicie Tuxedo:

```
$ tmboot -y
```

Qué hacer a continuación

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

AIX *Compilación del entorno de servidor en AIX (64 bits)*

Cómo compilar el entorno de servidor en IBM MQ for AIX (64 bits).

Procedimiento

1. Cree un directorio (por ejemplo, APPDIR) en el que se compile el entorno de servidor y ejecute todos los comandos en ese directorio.
2. Exporte las variables de entorno siguientes, donde TUXDIR representa el directorio raíz para TUXEDO, y *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el cual está instalado IBM MQ.

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Añada la línea siguiente al archivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Ejecute los comandos:

```
$ 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. Edite ubbstxcx.cfg y añada los detalles del nombre de máquina, los directorios de trabajo y el gestor de colas según sea necesario:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Cree el TLOGDEVICE:

```
$tmadmin -c
```

   Aparece un símbolo del sistema. En dicho símbolo, especifique:

```
> crdl -z /APPDIR/TLOG1
```

7. Inicie el gestor de colas:

```
$ strmqm
```

8. Inicie Tuxedo:

```
$ tmboot -y
```

Qué hacer a continuación

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Windows *Compilación del entorno de servidor en Windows (32 bits)*
Compilación del entorno de servidor en IBM MQ for Windows (32 bits).

Acerca de esta tarea

Nota: Cambie los siguientes campos identificados como *VARIABLES* a las rutas de directorio:

Tabla 164. Campos que hay que cambiar a rutas de directorio

Campo	**Ruta del directorio**
MQMDIR	Ruta de directorio especificada al instalarse IBM MQ, por ejemplo, `g:\Program Files\IBM\MQ`.
TUXDIR	La ruta de directorio especificada al instalarse TUXEDO, por ejemplo, `f:\tuxedo`.
APPDIR	Ruta de directorio que se va a utilizar en la aplicación de ejemplo, por ejemplo `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 134. Ejemplo de archivo ubbstxcn.cfg para IBM MQ for Windows

Nota: Cambie el nombre de la máquina *NombreMaquina* y las rutas de directorio para que coincidan con las de la instalación. Cambie también el nombre del gestor de colas *MIGESTORCOLAS* por el nombre del gestor de colas con el que desee conectarse.

El archivo de ejemplo `ubbconfig` de IBM MQ for Windows se lista en Figura 134 en la página 1157. Se suministra como `ubbstxcn.cfg` el el directorio de ejemplos de IBM MQ.

El archivo make de ejemplo (consulte Figura 135 en la página 1158) proporcionado para IBM MQ for Windows se denomina `ubbstxmn.mak`, y se mantiene en el directorio de ejemplos de IBM MQ.

```
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
```

Figura 135. Archivo make de TUXEDO de ejemplo para IBM MQ for Windows

Para compilar el entorno de servidor y los ejemplos, siga los pasos siguientes.

Procedimiento

1. Cree un directorio de aplicación en el que compilar la aplicación de ejemplo, por ejemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie los archivos de ejemplo siguientes del directorio de ejemplos de IBM MQ en el directorio de la aplicación:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Edite cada uno de estos archivos para configurar los nombres y rutas de directorio usados en la instalación.
4. Edite `ubbstxcn.cfg` (consulte Figura 134 en la página 1157) para añadir detalles del nombre de máquina y el gestor de colas al que desea conectarse.
5. Añada la línea siguiente al archivo TUXEDO *TUXDIR*`udataobj\rm`:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   La nueva entrada tiene que ocupar una única línea en el archivo.
6. Defina las variables de entorno siguientes:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Cree un dispositivo TLOG para TUXEDO.

   Para ello, invoque `tmadmin -c` y especifique el comando siguiente:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Establezca el directorio actual a *APPDIR* e invoque el archivo make de ejemplo `amqstxmn.mak` como un archivo make de proyecto externo. Por ejemplo, con Microsoft Visual C++, emita el mandato siguiente:

```
msvc amqstxmn.mak
```

   Seleccione **compilar** para compilar todos los programas de ejemplo.

Windows *Compilación del entorno de servidor en Windows (64 bits)*

Cómo compilar el entorno de servidor en IBM MQ for Windows (64 bits).

Acerca de esta tarea

Nota: Cambie los siguientes campos identificados como *VARIABLES* a las rutas de directorio:

Tabla 165. Campos que hay que cambiar a rutas de directorio

Campo	**Ruta del directorio**
MQMDIR	Ruta de directorio especificada al instalarse IBM MQ, por ejemplo, `g:\Program Files\IBM\MQ`.
TUXDIR	La ruta de directorio especificada al instalarse TUXEDO, por ejemplo, `f:\tuxedo`.
APPDIR	Ruta de directorio que se va a utilizar en la aplicación de ejemplo, por ejemplo `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figura 136. Ejemplo de archivo ubbstxcn.cfg para IBM MQ for Windows

Nota: Cambie el nombre de la máquina *NombreMaquina* y las rutas de directorio para que coincidan con las de la instalación. Cambie también el nombre del gestor de colas *MIGESTORCOLAS* por el nombre del gestor de colas con el que desee conectarse.

El archivo de ejemplo ubbconfig o para IBM MQ for Windows aparece listado en Figura 136 en la página 1160. Se suministra como `ubbstxcn.cfg` el el directorio de ejemplos de IBM MQ.

El archivo make de ejemplo (consulte Figura 137 en la página 1161) proporcionado para IBM MQ for Windows se denomina `ubbstxmn.mak`, y se mantiene en el directorio de ejemplos de IBM MQ.

```
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
```

Figura 137. Archivo make de TUXEDO de ejemplo para IBM MQ for Windows

Para compilar el entorno de servidor y los ejemplos, siga los pasos siguientes.

Procedimiento

1. Cree un directorio de aplicación en el que compilar la aplicación de ejemplo, por ejemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie los archivos de ejemplo siguientes del directorio de ejemplos de IBM MQ en el directorio de la aplicación:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Edite cada uno de estos archivos para configurar los nombres y rutas de directorio usados en la instalación.
4. Edite `ubbstxcn.cfg` (consulte Figura 136 en la página 1160) para añadir detalles del nombre de máquina y el gestor de colas al que desea conectarse.
5. Añada la línea siguiente al archivo TUXEDO *TUXDIR*`udataobj\rm`

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   La nueva entrada tiene que ocupar una única línea en el archivo.
6. Defina las variables de entorno siguientes:

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Cree un dispositivo TLOG para TUXEDO. Para ello, invoque `tmadmin -c` y especifique el comando siguiente:

```
crdl -z APPDIR\TLOG
```

8. Establezca el directorio actual a *APPDIR* e invoque el archivo make de ejemplo `amqstxmn.mak` como un archivo make de proyecto externo. Por ejemplo, con Microsoft Visual C++, emita el mandato siguiente:

   ```
   msvc amqstxmn.mak
   ```

   Seleccione **compilar** para compilar todos los programas de ejemplo.

ALW *Programa servidor de ejemplo para TUXEDO*

El programa servidor de ejemplo (amqstxsx) se ha diseñado para ejecutar con los programas de ejemplo de colocación (amqstxpx.c) y obtención (amqstxgx.c). El programa servidor de ejemplo ejecuta de forma automática cuando se inicia TUXEDO.

Nota: Hay que iniciar el gestor de colas antes de iniciar TUXEDO.

El servidor de ejemplo proporciona dos servicios TUXEDO, MPUT1 y MGET1:

- El servicio MPUT1 está controlado por el ejemplo PUT y utiliza MQPUT1 en punto de sincronización para colocar un mensaje en una unidad de trabajo controlada por TUXEDO. Recibe los parámetros QName (nombre de cola) y Message Text (mensaje de texto), proporcionados por el ejemplo PUT.
- Cada vez que recibe un mensaje, el servicio MGET1 abre y cierra la cola. Recibe los parámetros QName (nombre de cola) y Message Text (mensaje de texto), proporcionados por el ejemplo GET.

Los mensajes de error, los códigos de razón y los mensajes de estado se escriben en el archivo de registro cronológico de TUXEDO.

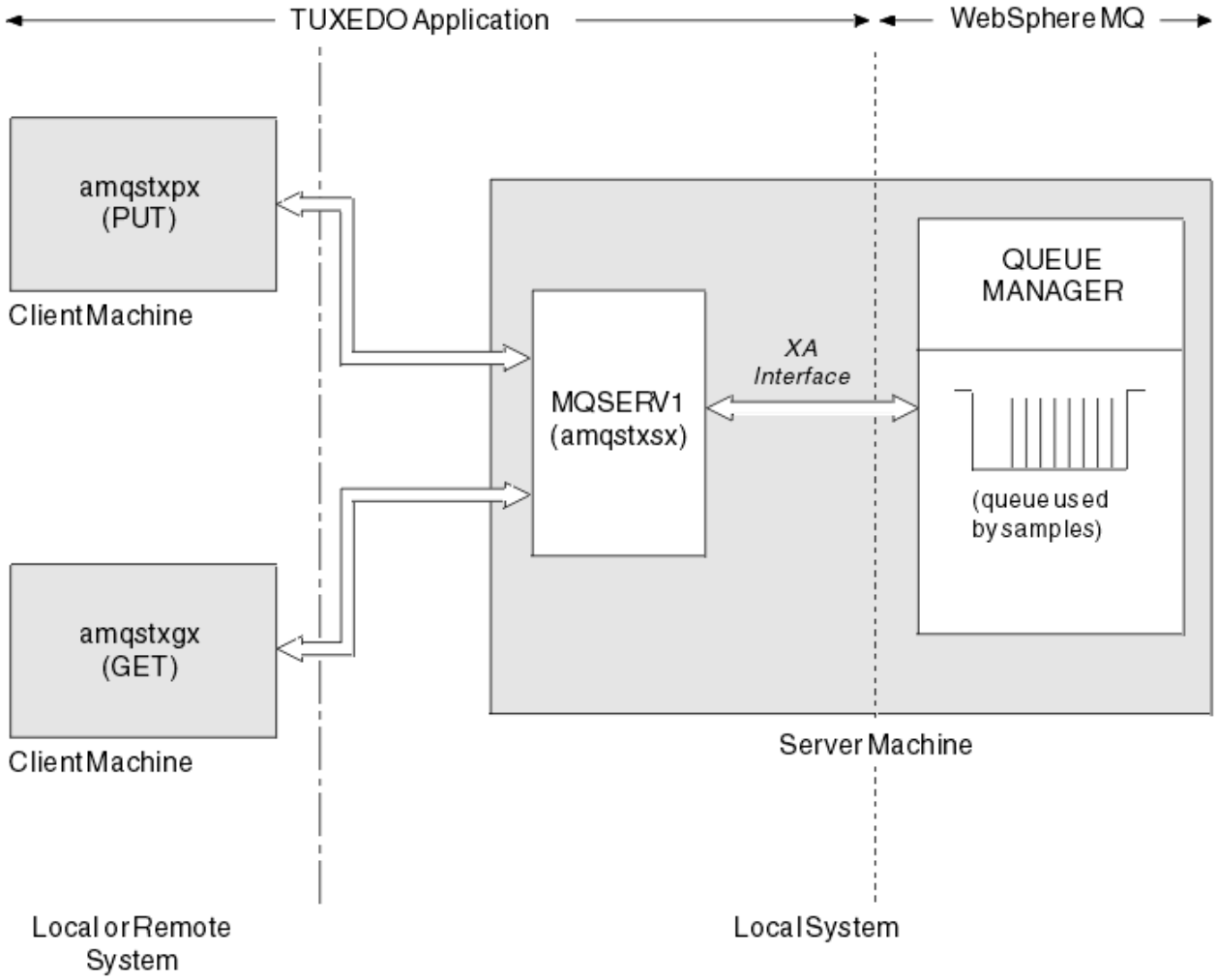


Figura 138. Cómo funcionan juntos los ejemplos de TUXEDO

ALW *Programa de ejemplo de Put para TUXEDO*

Este ejemplo le permite poner un mensaje en una cola varias veces, en lotes, mostrando la sincronización y usando TUXEDO como el gestor de recursos.

El programa de servidor de ejemplo `amqstxsx` debe estar en ejecución para que el ejemplo de colocación sea satisfactorio; el programa de ejemplo de servidor se conecta al gestor de colas y utiliza la interfaz XA. Para ejecutar el ejemplo, especifique:

- `doputs -n queuename -b batchsize -c trancount -t message`

Por ejemplo:

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Así se colocan 30 mensajes en la cola `myqueue`, en seis lotes de cinco mensajes cada uno. Si hay algún problema, se retiene un lote de mensajes; de lo contrario, se confirma.

Los mensajes de error se escriben en el archivo de registro de TUXEDO y en la salida de error estándar (stderr). Los códigos de razón se escriben en stderr.

ALW *Ejemplo de Get para TUXEDO*

Este ejemplo le permite obtener mensajes de una cola en lotes.

El programa de servidor de ejemplo `amqstxsx` debe estar en ejecución para que el ejemplo Get sea satisfactorio; el programa de servidor de ejemplo se conecta al gestor de colas y utiliza la interfaz XA. Para ejecutar el ejemplo, especifique el mandato siguiente:

- `dogets -n queuename -b batchsize -c trancount`

Por ejemplo:

- `dogets -n myqueue -b 6 -c 4`

Retira 24 mensajes en la cola `myqueue`, en seis lotes de cuatro mensajes cada uno. Si lo ejecuta después del ejemplo de put, que colocaba 30 mensajes en `myqueue`, solo quedarán seis mensajes en `myqueue`. El número de lotes y su tamaño puede variar entre la colocación de los mensajes y su retirada.

Los mensajes de error se escriben en el archivo de registro de TUXEDO y en la salida de error estándar (stderr). Los códigos de razón se escriben en stderr.

Windows *Utilización de la salida de seguridad SSPI en Windows*

En este tema se describe cómo utilizar los programas de salida de canal SSPI en sistemas Windows. El código de salida suministrado tiene dos formatos: el objeto y origen.

Código de objeto

El archivo de código de objeto se denomina `amqrspin.dll`. Para el cliente y el servidor, se instala como una parte estándar de IBM MQ for Windows en la carpeta *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME*. Por ejemplo, `C:\Archivos de programa\IBM\MQ\exits\installation2`. Se carga como una salida de usuario estándar. Puede ejecutar la salida de canal de seguridad proporcionada y utilizar los servicios de autenticación en la definición del canal.

Para ello, especifique cualquiera de los valores siguientes:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Para proporcionar soporte para un canal restringido, especifique lo siguiente en el canal SVRCONN:

```
SCYDATA('remote_principal_name')
```

donde *nombre_principal_remoto* tiene el formato *DOMINIO**usuario*. El canal seguro sólo se establece si el nombre del principal remoto coincide con el valor *nombre_principal_remoto*.

Para utilizar los programas de salida de canal proporcionados entre los sistemas que operen dentro de un dominio de seguridad Kerberos, cree un valor **servicePrincipalName** para el gestor de colas.

Código fuente

El archivo de código fuente de salida se llama `amqsspin.c`. Está en `C:\Archivos de programa\IBM\MQ\Tools\c\Samples`.

Si modifica el código fuente, deberá recompilar la parte que se haya modificado.

Se compila y se enlaza del mismo modo que cualquier otra de salida de canal para la plataforma pertinente, excepto que se debe acceder a las cabeceras SSPI en el momento de la compilación, y que se debe acceder a las bibliotecas de seguridad SSPI, junto con todas las bibliotecas asociadas recomendadas, en el momento de efectuar el enlace.

Antes de ejecutar el mandato siguiente, asegúrese de que `cl.exe` y la biblioteca de Visual C++ y la carpeta `include` estén disponibles en la vía de acceso. Por ejemplo:

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

Nota: El código fuente no incluye ninguna sección para rastrear ni manejar los errores. Si modifica y utiliza el código fuente, añada sus propias rutinas de rastreo y de manejo de errores.

Ejecución de los ejemplos utilizando colas remotas

Se puede ilustrar la gestión de colas remotas ejecutando los ejemplos en gestores de colas conectados.

El programa amqscos0.tst proporciona una definición local de una cola remota (SYSTEM.SAMPLE.REMOTE) que utiliza un gestor de colas remoto llamado OTHER. Para usar esta definición, cambie OTHER al nombre del segundo gestor de colas que se desee usar. También hay que configurar un canal de mensajes entre ambos gestores de colas; para obtener información sobre cómo hacer esto, consulte Definición de los canales.

Los programas de ejemplo de solicitud colocan su propio nombre de gestor de colas local en el campo *ReplyToQMgr* de mensajes que envían. Los ejemplos de consulta y establecimiento envían mensajes de respuesta a la cola y al gestor de colas de mensajes denominados en los campos *ReplyToQ* y *ReplyToQMgr* de los mensajes de solicitud que procesan.

El programa de ejemplo Cluster Queue Monitoring (AMQSCLM)

Este ejemplo utiliza las características de equilibrio de carga de trabajo de IBM MQ para dirigir mensajes a instancias de colas que tienen conectadas aplicaciones consumidoras. Esta dirección automática impide la acumulación de mensajes en una instancia de una cola de clúster a la que no está conectada ninguna aplicación consumidora.

Visión general

Puede configurar un clúster que tiene más de una definición para la misma cola en diferentes gestores de colas. Esta configuración proporciona la ventaja de una disponibilidad y un equilibro de carga de trabajo mejorados. No obstante, no hay ninguna prestación incorporada en IBM MQ para modificar de forma dinámica la distribución de mensajes por un clúster basándose en el estado de aplicaciones conectadas. Por este motivo, una aplicación consumidora debe estar siempre conectada a cada instancia de una cola para garantizar que se procesen mensajes.

El programa de ejemplo de supervisión de cola de clúster supervisa el estado de aplicaciones conectadas. El programa ajusta dinámicamente la configuración de equilibrio de carga incorporado para dirigir mensajes a instancias de una cola en clúster con aplicaciones consumidoras conectadas. En determinadas situaciones, este programa se puede utilizar para disminuir la necesidad de que una aplicación consumidora esté siempre conectada a cada instancia de una cola. También vuelve a enviar mensajes que están en cola en una instancia de una cola sin aplicaciones consumidoras conectadas. El reenvío de mensajes permite que los mensajes se redirijan alrededor de una aplicación consumidora que está apagada temporalmente.

El programa está diseñado para utilizarse en el caso de las aplicaciones consumidoras sean de larga ejecución, en lugar de aplicaciones que se conectan y desconectan con frecuencia.

El programa de ejemplo de supervisión de cola en clúster es el programa ejecutable compilado del archivo de ejemplo C `amqsclma.c`.

Se puede encontrar información adicional sobre clústeres y la carga de trabajo en Utilización de clústeres para la gestión de carga de trabajo

AMQSCLM: Diseño y planificación para usar el ejemplo

Información sobre cómo funciona el programa de ejemplo de supervisión de colas de clúster, puntos a tener en cuenta al configurar un sistema para que ejecute el programa de ejemplo y modificaciones que se pueden realizar en el código fuente de ejemplo.

Diseño

El programa de ejemplo de supervisión de colas de clúster supervisa colas de clúster locales que tienen aplicaciones consumidoras conectadas. El programa supervisa las colas especificadas por el usuario. El nombre de la cola puede ser específico, por ejemplo `APP.TEST01`, o genérico. Los nombres genéricos han de tener un formato que se ajuste al formato de comandos programables (Programmable Command Format, PCF). Algunos ejemplos de nombres genéricos son `APP.TEST*` o `APP*`.

Cada gestor de colas de clúster que sea propietario de una instancia de una cola local que haya que supervisar, requiere que se le conecte una instancia del programa de ejemplo de supervisión de colas de clúster.

Direccionamiento de mensajes dinámico

El programa de ejemplo de supervisión de colas de clúster utiliza el valor **IPPROCS** (abierta para el recuento de procesos de entrada) de una cola para determinar si esa cola tiene algún consumidor. Un valor mayor que 0 indica que la cola tiene conectada al menos una aplicación consumidora. Tales colas están activas. Un valor de 0 indica que la cola no tiene ningún programa de consumidor conectado. Tales colas están inactivas.

En una cola de clúster con múltiples instancias en clúster, IBM MQ usa la propiedad de prioridad de carga de trabajo de clúster **CLWLPRTY** de cada instancia de cola para determinar a qué instancias enviar los mensajes. IBM MQ envía mensajes a las instancias disponibles de una cola con el valor **CLWLPRTY** más alto.

El programa de ejemplo de supervisión de colas de clúster activa una cola de clúster estableciendo el valor local de **CLWLPRTY** en 1. El programa desactiva una cola de clúster estableciendo su valor **CLWLPRTY** en 0.

La tecnología de la agrupación en clúster de IBM MQ propaga la propiedad **CLWLPRTY** actualizada de una cola en clúster a todos los gestores de colas relevantes del clúster. Por ejemplo,

- Un gestor de colas con una aplicación conectada que coloca mensajes en la cola.
- Un gestor de colas propietario de una cola local del mismo nombre en el mismo clúster.

La propagación se realiza utilizando los gestores de colas de repositorio completo del clúster. Los mensajes nuevos de la cola de clúster se dirigirán a las instancias que tengan el valor **CLWLPRTY** más alto dentro del clúster.

Transferencia de mensajes en cola

La modificación dinámica del valor de **CLWLPRTY** influye en el direccionamiento de los mensajes nuevos. Esta modificación dinámica no afecta a los mensajes que ya están encolados en una instancia de cola sin consumidores conectados ni a los mensajes que ya habían pasado por el mecanismo de equilibrado de cargas de trabajo antes de que se propagara por el clúster un valor de **CLWLPRTY** modificado. Por tanto, los mensajes permanecerán en cualquier cola inactiva y no serán procesados por niniguna aplicación consumidora. Para solucionar esto, el programa de ejemplo de supervisión de colas de clúster puede obtener mensajes de una cola local sin consumidores y enviar dichos mensajes a instancias remotas de la misma cola a la que están conectados los consumidores.

El programa de ejemplo de supervisión de colas de clúster transfiere mensajes de una cola local inactiva a una o más colas remotas activas obteniendo mensajes (con **MQGET**) y colocando mensajes (con **MQPUT**) en la misma cola de clúster. Esta transferencia provoca que la gestión de equilibrado de cargas de trabajo de IBM MQ seleccione una instancia de destino distinta basándose en un valor CLWLPRTY más alto que el de la instancia de cola loca. La persistencia de mensajes y el contexto se conservan durante la transferencia de mensajes. El orden de los mensajes y las opciones de enlace no se conservan.

Planificación

El programa de ejemplo de supervisión de colas de clúster modifica la configuración del clúster cuando se produce un cambio en la conectividad de las aplicaciones consumidoras. Las modificaciones se transmiten desde los gestores de colas en los que el programa de ejemplo de supervisión de colas de clúster está supervisando colas a los gestores de colas de repositorio completo del clúster. Los gestores de colas de repositorio completo procesan las actualizaciones de configuración y las reenvían a todos los gestores de colas relevantes del clúster. Los gestores de colas relevantes incluyen los gestores de colas que poseen colas agrupadas en clúster del mismo nombre (donde se ejecuta una instancia del programa de ejemplo de supervisión de colas de clúster) y cualquier gestor de colas en el que una aplicación ha abierto la cola de clúster para colocarle mensajes en los últimos 30 días.

Los cambios se procesan de forma asíncrona en el clúster. Por tanto, tras cada cambio, los distintos gestores de colas del clúster podrían tener diferentes vistas de la configuración durante cierto tiempo.

El programa de ejemplo de supervisión de colas en clúster solo es adecuado en sistemas donde las aplicaciones consumidoras se conectan o desconectan con poca frecuencia; por ejemplo, aplicaciones consumidoras de larga ejecución. Cuando se utiliza para supervisar sistemas en los que solo se conectan aplicaciones consumidoras durante periodos cortos, la latencia en que se incurre al distribuir las actualizaciones de configuración podría dar lugar a que los gestores de colas del clúster tenga una vista incorrecta de las colas a las que están conectados las consumidoras. Esta latencia puede provocar un direccionamiento incorrecto de mensajes.

Cuando se supervisan muchas colas, una tasa relativamente baja de cambios en los consumidores conectados en todas las colas podría aumentar el tráfico de configuración en el clúster. El aumento del tráfico de configuración de clúster puede provocar una carga excesiva en uno o más de los gestores de colas siguientes:

- Los gestores de colas donde ejecuta el programa de ejemplo de supervisión de colas de clúster.
- Los gestores de colas del repositorio completo.
- Un gestor de colas con una aplicación conectada que coloca mensajes en la cola.
- Un gestor de colas propietario de una cola local del mismo nombre en el mismo clúster.

Hay que evaluar el uso de procesador en los gestores de colas de repositorio completo. El uso del procesador adicional se visualiza en el tráfico de mensajes en la cola de repositorio completo SYSTEM.CLUSTER.COMMAND.QUEUE. Si se crean mensajes en dicha cola, es síntoma de que los gestores de colas de repositorio completo no pueden seguir el ritmo de cambios de configuración de clúster en el sistema.

Cuando el programa de ejemplo de supervisión de colas en clúster está supervisando muchas colas, el programa de ejemplo y el gestor de colas realizan cierta cantidad de trabajo. Dicho trabajo se lleva a cabo incluso cuando no hay cambios en los consumidores conectados. Se puede modificar el argumento **-i** para reducir el uso de procesador del programa de ejemplo en el sistema local al reducir la frecuencia de ciclos de supervisión.

Para ayudar a detectar una actividad excesiva, el programa de ejemplo de supervisión de colas de clúster notifica el tiempo de procesamiento promedio por intervalo de sondeo, el tiempo de procesamiento transcurrido y el número de cambios de configuración. Los informes se entregan en un mensaje informativo, **CLM0045I**, cada 30 minutos, o cada 600 intervalos de sondeo, lo que ocurra antes.

Requisitos de uso de la supervisión de colas de clúster

El programa de ejemplo de supervisión de colas de clúster tiene requisitos y restricciones. Se puede modificar el código fuente del ejemplo proporcionado para cambiar algunas de estas restricciones de uso. En los ejemplos listados en esta sección se detallan las modificaciones que se pueden efectuar.

- El programa de ejemplo de supervisión de colas de clúster se ha diseñado para supervisar las colas a las que las aplicaciones consumidoras están conectadas o no. Si el sistema tiene aplicaciones consumidoras que suelen conectar y desconectar, el programa de ejemplo puede generar una actividad de configuración excesiva en todo el clúster. Esto podría penalizar el rendimiento de los gestores de colas del clúster.
- El programa de ejemplo de supervisión de colas de clúster depende del sistema IBM MQ subyacente y la tecnología de clúster. El número de colas supervisadas, la frecuencia de supervisión y la frecuencia de cambios de estado de cada cola afectan a la carga del sistema global. Estos factores se deben tener en cuenta al seleccionar las colas que se van a supervisar y el intervalo de sondeo de la supervisión.
- Tiene que haber una instancia del programa de ejemplo de supervisión de colas de clúster conectada con cada gestor de colas del clúster que posea una instancia de una cola supervisada. No es necesario conectar el programa de ejemplo a gestores de colas del clúster que no posean las colas.
- El programa de ejemplo de supervisión de colas de clúster se debe ejecutar con la debida autorización para acceder a todos los recursos de IBM MQ necesarios. Por ejemplo,
  - El gestor de colas al que se va a conectar.

  - SYSTEM.ADMIN.COMMAND.QUEUE.
  - Todas las colas que haya que supervisar cuando se realiza la transferencia de mensajes.
- El servidor de comandos tiene que estar ejecutando para cada gestor de colas que tenga conectado el programa de ejemplo de supervisión de colas de clúster.
- Cada instancia del programa de ejemplo de supervisión de colas de clúster requiere un uso exclusivo de una cola local (no agrupada en clúster) en el gestor de colas al que está conectado. Dicha cola local se utiliza para controlar el programa de ejemplo y recibir mensajes de respuesta de las consultas realizadas en el servidor de comandos del gestor de colas.
- Todas las colas que tengan que supervisarse mediante una única instancia del programa de ejemplo de supervisión de colas de clúster habrán de estar en el mismo clúster. Si un gestor de colas tiene colas en varios clústeres que requieren supervisión, se necesitarán varias instancias del programa de ejemplo. Cada instancia necesita una cola local para los mensajes de control y respuesta.
- Todas las colas que haya que supervisar han de estar en un único clúster. Las colas configuradas para utilizar una lista de nombres de clúster no se supervisan.
- La habilitación de la transferencia de mensajes procedentes de colas inactivas es opcional. Se aplica a todas las colas supervisadas por las instancias del programa de ejemplo de supervisión de colas de clúster. Si solo un subconjunto de las colas supervisadas requieren tener habilitada la transferencia e mensajes, se necesitarán dos instancias del programa de ejemplo de supervisión de colas de clúster. Un programa de ejemplo tendrá habilitada la transferencia de mensajes y el otro la tendrá inhabilitada. Cada instancia del programa de ejemplo necesita una cola local para los mensajes de control y respuesta.
- De forma predeterminada, el equilibrio de carga de trabajo de clúster de IBM MQ enviará mensajes a instancias de colas en clúster que residen en el mismo gestor de colas al que está conectada una aplicación que coloca. Esto tendrá que inhabilitarse mientras la cola local esté inactiva en los casos siguientes:
  - Las aplicaciones que colocan se conectan con gestores de colas que poseen instancias de una cola inactiva y que se están supervisando.
  - Los mensajes encolados se transfieren de colas inactivas a colas activas.

  La preferencia de equilibrado de cargas de trabajo local en la cola se puede inhabilitar estáticamente, estableciendo el valor `CLWLUSEQ` a `ANY`. En esta configuración, los mensajes colocados en colas locales se distribuyen a instancias de colas colas locales y remotas para equilibrar la carga de trabajo, incluso cuando hay aplicaciones consumidoras locales. De forma alternativa, el programa de ejemplo de supervisión de colas de clúster se puede configurar para establecer temporalmente el valor **CLWLUSEQ** a `ANY` mientras la cola no tenga consumidores conectados, lo que da lugar a que solo mensajes locales vayan a instancias locales de una cola mientras dicha cola esté activa.
- El sistema y las aplicaciones IBM MQ no deben utilizar **CLWLPRTY** para las colas que se van a supervisar ni para los canales que se están utilizando. De lo contrario, las acciones del programa de ejemplo de supervisión de colas de clúster sobre los atributos de cola **CLWLPRTY** podrían tener efectos indeseados.
- El programa de ejemplo de supervisión de colas de clúster anota información en tiempo de ejecución en un conjunto de archivos de informe. Se necesita un directorio para almacenar dichos informes y el programa de ejemplo de supervisión de colas de clúster habrá de tener autorización para escribir en el.

AMQSCLM: Preparación y ejecución del ejemplo
El ejemplo de supervisión de la cola de clúster se puede ejecutar en local o conectado a un gestor de cola, o como un cliente conectado sobre un canal. El ejemplo debe estar en ejecución siempre que se esté ejecutando el gestor de colas; cuando se ejecute localmente, se puede configurar como un servicio de gestor de colas para que el ejemplo se inicie automáticamente y se detenga con el gestor de colas.

Antes de empezar

Los pasos siguientes se deben completar antes de ejecutar el ejemplo de supervisión de colas de clúster.

1. Cree una cola de trabajo en cada gestor de colas para el uso interno del ejemplo.

Cada instancia del ejemplo necesita una cola local que no sea de clúster para uso interno exclusivo. Puede elegir el nombre de la cola. En el ejemplo se utiliza el nombre `AMQSCLM.CONTROL.QUEUE`. Por ejemplo, en Windows, puede crear esta cola utilizando el siguiente mandato **MQSC**:

```
DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
```

Puede dejar los valores de **MAXDEPTH** y **MAXMSGL** como valor predeterminado.

2. Cree un directorio para los registros de mensajes de error e informativos.

   El ejemplo graba mensajes de diagnóstico en los archivos de informe. Debe elegir un directorio en el que almacenar los archivos. Por ejemplo, en Windows, puede crear un directorio utilizando el mandato siguiente:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   Los archivos de informe creados por el ejemplo tienen el convenio de nomenclatura siguiente:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (Opcional) Defina el ejemplo de supervisión de colas de clúster como un servicio de IBM MQ.

   Para supervisar las colas, el ejemplo siempre debe estar en ejecución. Para asegurarse de que el ejemplo de supervisión de colas de clúster siempre esté en ejecución, puede definir el ejemplo como un servicio de gestor de colas. La definición del ejemplo como servicio significa que AMQSCLM se inicia cuando se inicia el gestor de colas. Puede utilizar el ejemplo siguiente para definir el ejemplo de supervisión de colas de clúster como un servicio de IBM MQ.

   ```
   define service(AMQSCLM) +
         descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
         control(qmgr) +
         servtype(server) +
         startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
         startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
   c:\AMQSCLM\rpts') +
         stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
         stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

Definición	Descripción
service	Especifica el nombre de servicio. Puede elegir el nombre de servicio.
descr	Especifica una descripción de texto del servicio.
control	Indica que el servicio se inicia y se detiene al mismo tiempo que el gestor de colas.
servtype	Indica que un objeto de servicio de servidor, que significa que sólo una instancia se puede ejecutar a la vez para este gestor de colas.
startcmd	Especifica la ubicación y el nombre del programa.
startarg	Especifica los argumentos del ejemplo. Tenga en cuenta el uso de *+ QMNAME +*. El nombre del gestor de colas se sustituye automáticamente.
stdout	El nombre de archivo totalmente calificado al que se redirige la salida estándar. El ejemplo graba en este archivo sólo los mensajes que confirman que la muestra ha terminado. El ejemplo hace esto porque el archivo de error estándar ya se ha cerrado en una etapa anterior del proceso de terminación de ejemplo.
stderr	El nombre de archivo totalmente calificado al que se redirige la salida de error estándar. El ejemplo graba en el archivo de error estándar cualquier mensaje de error antes de la terminación de la muestra.

Acerca de esta tarea

Esta tarea le permite iniciar y detener el ejemplo de supervisión de colas de clúster de diferentes maneras. También le permite ejecutar el ejemplo en una modalidad que genera archivos de informe que contienen información estadísticas sobre las colas que se supervisan.

El programa de ejemplo se puede ejecutar utilizando el mandato siguiente.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

En la tabla se listan los argumentos que se pueden utilizar con el ejemplo de supervisión de colas de clúster, junto con información adicional sobre cada uno.

Argumento	Variable	Información complementaria
-m	**QMgrName**	El gestor de colas que se va a supervisar.
-c	**ClusterName**	El clúster que contiene las colas que se van a supervisar.
-q	**QNameMask**	La cola, o las colas, para supervisar. Un * al final supervisa todas las colas con nombres que coinciden con cero o más caracteres al final.
-f	**QListFile**	La vía de acceso completa y el nombre de archivo de un archivo que contiene una lista de nombres de cola o máscaras de nombre de cola que se deben supervisar. El archivo debe contener un nombre de cola/ máscara por línea. Puede especificar **-q** o **-f**, pero no ambos.
-r	**MonitorQName**	La cola local que está siendo utilizada exclusivamente por el ejemplo.
-l	**ReportDir**	La vía de acceso del directorio en el que se almacenan los mensajes de información registrados en un conjunto de acomodaciones[9]Archivos de informes.
-t		(Opcional) Habilita la transferencia de mensajes en cola de las colas locales inactivas a las colas activas. Si no está habilitado, sólo los mensajes nuevos que entren en el clúster se direccionan dinámicamente a instancias activas de una cola.
-u	**ActiveVal**	(Opcional) Conmuta automáticamente la propiedad **CLWLUSEQ** de una instancia de cola supervisada a ANY cuando está inactiva, y al valor de **ActiveVal** cuando está activa. **ActiveVal** puede ser LOCAL o QMGR. Si este argumento no se establece en un sistema en el que las aplicaciones se conectan al mismo gestor de colas, o donde está habilitada la transferencia de mensajes, las colas supervisadas deben tener un valor **CLWLUSEQ** de ANY o QMGR, teniendo el gestor de colas el valor ANY.
-i	**Interval**	(Opcional) El intervalo de tiempo en segundos, en el que el supervisor comprueba las colas. El valor predeterminado es de 300 segundos (5 minutos).
-d		(Opcional) Habilita la salida de diagnóstico adicional. Puede ser útil la depuración de la salida en la configuración inicial del sistema, o cuando se trabaja con el código de ejemplo.
-s		(Opcional) Habilita la salida estadística mínima por intervalo.
-v		(Opcional) Registre la información del informe en `standard out`, además de los archivos de informe.

[9] Para cada combinación de gestor de colas y cola se genera un archivo de registro de tamaño fijo que, cuando está lleno, se sobrescribe. El registrador siempre escribe en el mismo archivo, y también mantiene las dos versiones anteriores del archivo.

Ejemplos de la lista de argumentos:

```
-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
```

Archivo de lista de colas de ejemplo:

```
Q1
QUEUE.*
ABC
ABD
```

Procedimiento

1. Inicie el ejemplo de supervisión de colas de clúster. Puede iniciar el ejemplo de una de las maneras siguientes:
   - Utilice un indicador de mandatos con las autorizaciones de usuario adecuadas.
   - Utilice el mandato MQSC **START SERVICE**, si el ejemplo está configurado como un servicio de IBM MQ.

   La lista de argumentos es la misma en ambos casos.

   El ejemplo no inicia la supervisión de las colas durante 10 segundos después de que se inicialice el programa. Este retardo permite que las aplicaciones consumidoras se conecten primero a las colas supervisadas, evitando cambios innecesarios en el estado activo de la cola.
2. Detenga el ejemplo de supervisión de colas de clúster. El ejemplo se detiene automáticamente cuando el gestor de colas se detiene, se está deteniendo o está pasando a inactivo, o si se rompe la conexión con el gestor de colas. Hay formas de detener el ejemplo sin finalizar el gestor de colas:
   - Configure la cola local utilizada exclusivamente por el ejemplo para inhabilitar la función Get.
   - Envíe un mensaje con un **CorrelId** de `"STOP CLUSTER MONITOR\0\0\0\0"`, a la cola local utilizada exclusivamente por el ejemplo.
   - Termine el proceso de ejemplo. Esto puede dar como resultado la pérdida de mensajes no persistentes que se transfieren a las colas activas. También puede provocar que la cola local utilizada por el ejemplo se mantenga abierta durante unos segundos después de la terminación. Esta situación evita que se inicie de forma inmediata una nueva instancia del ejemplo de supervisión de colas de clúster.

   Si el ejemplo se ha iniciado como un servicio IBM MQ, **STOP SERVICE** no tiene efecto. Se puede utilizar uno de los métodos de terminación descritos como un mecanismo de **STOP SERVICE** configurado en el gestor de colas.

Qué hacer a continuación

Compruebe el estado del ejemplo.

Si la creación de informes está habilitada, puede revisar los archivos de informe para ver el estado. Utilice el mandato siguiente para revisar el archivo de informe más actual:

```
QMgrName.ClusterName.RPT01.LOG
```

Para revisar los archivos de informe más antiguos, utilice los mandatos siguientes:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Los archivos de informe crecen a un tamaño máximo de 1 MB aproximadamente. Cuando se llena el archivo RPT01, se crea un nuevo archivo RPT01. El antiguo archivo RPT01 se renombra como RPT02. RPT02 se renombra a RPT03. El RPT03 antiguo se descarta.

El ejemplo crea mensajes de información en las situaciones siguientes:

- durante el inicio
- al terminar
- cuando marca una cola **ACTIVE** o **INACTIVE**
- cuando vuelve a poner en cola los mensajes de una cola inactiva a una instancia o instancias activas

El ejemplo crea un mensaje de error *CLMnnnnE* para informar de un problema que requiere atención.

Cada 30 minutos, el ejemplo muestra el promedio de tiempo de proceso por intervalo de sondeo y el tiempo de proceso transcurrido. Esta información se incluye en el mensaje CLM0045I.

Cuando se habilitan mensajes estadísticos **-s**, el ejemplo proporciona la siguiente información estadística sobre cada comprobación de cola:

- Tiempo que se ha tardado en procesar las colas (en milisegundos)
- Cantidad de colas comprobadas
- Cambios activos/inactivos realizados
- Cantidad de mensajes transferidos

Esta información se notifica en el mensaje CLM0048I.

Los archivos de informe pueden crecer rápidamente en modalidad de depuración y autoajustarse rápidamente. En esta situación, es posible que se sobrepase el límite de tamaño de 1 MB para archivos individuales.

AMQSCLM: Resolución de problemas
Las secciones siguientes contienen información sobre escenarios que se pueden dar al utilizar el ejemplo. Se proporciona información sobre las explicaciones potenciales de un escenario y las opciones sobre cómo resolverlo.

Escenario: AMQSCLM no arranca

Explicación potencial: Sintaxis incorrecta.

Acción: Compruebe la sintaxis correcta en la salida de error estándar

Explicación Posible: El gestor de colas no está disponible.

Acción: Compruebe el ID de mensaje CLM0010E en el archivo de informe.

Explicación potencial: No se puede abrir o crear el archivo o archivos de informe.

Acción: Compruebe la existencia de mensajes de error al inicializar en la salida de error estándar

Escenario: AMQSCLM no está cambiando una cola a ACTIVE o INACTIVE

Explicación potencial: La cola no está en la lista de colas por supervisar

Acción: Compruebe los valores de los parámetros **-q** y **-f**.

Explicación de potencial: La cola no es una cola local en el clúster correcto.

Acción: Compruebe que la cola sea local y que esté en el clúster correcto.

Explicación de potencial: AMQSCLM no se está ejecutando para este gestor de colas y clúster.

Acción: Inicie AMQSCLM para el gestor de colas y el clúster relevantes.

Explicación potencial: La cola se deja INACTIVE, **CLWLPRTY** =0, porque no tiene consumidores. De forma alternativa, se deja ACTIVE **CLWLPRTY** >=1, porque tiene al menos 1 consumidor.

Acción: Compruebe si hay aplicaciones consumidoras conectadas a la cola.

Explicación de potencial: El servidor de comandos del gestor de colas no está ejecutando.

Acción: Compruebe si hay errores en los archivos de informe.

Escenario: Los mensajes no se direccionan por colas INACTIVE

Explicación potencial: Los mensajes se colocan directamente en el gestor de colas propietario de la cola inactiva y el valor **CLWLUSEQ** de la cola no es ANY, y el argumento **-u** no se utiliza en AMQSCLM.

Acción: Compruebe el valor de **CLWLUSEQ** del gestor de colas relevante o asegúrese de que se utiliza el argumento **-u** en AMQSCLM.

Explicación potencial: No hay ninguna cola activa en ningún gestor de colas. La carga de mensajes se equilibra de forma equitativa entre todas las colas inactivas hasta que una cola pasa a estar activa.

Acción: Compruebe el estado de las colas en todos los gestores de colas.

Explicación potencial: Los mensajes se colocan en un gestor de colas del clúster distinto del gestor que posee la cola inactiva y el valor de 0 actualizado de **CLWLPRTY** no se propaga al gestor de colas de la aplicación colocadora.

Acción: Compruebe que están ejecutando los canales de clúster entre el gestor de colas supervisado y el gestor de colas de repositorio completo. Compruebe que estén ejecutando los canales entre el gestor de colas colocador y el gestor de colas de repositorio completo. Compruebe los registros de errores de los gestores de colas supervisado, colocador y de repositorio completo.

Explicación potencial: Las instancias de cola remota están activas (CLWLPRTY=1), pero los mensajes no se pueden direccionar a las instancias de cola porque el canal emisor de clúster del gestor de colas local no está ejecutando.

Acción: Compruebe el estado de los canales emisores de clúster del gestor de colas local al gestor (o gestores) de colas remotos con una instancia activa de la cola.

Escenario: AMQSCLM no transfiere mensajes procedentes de una cola inactiva

Explicación potencial: La transferencia de mensajes no está habilitada (**-t**).

Acción: Asegúrese de que la transferencia de mensajes está habilitada (**-t**).

Explicación potencial: La cola no está en la lista de colas que se van a supervisar.

Acción: Compruebe los valores de los parámetros **-q** y **-f**.

Explicación potencial: AMQSCLM no está ejecutando para este gestor de colas, ni para otros en el clúster que son propietarios de la misma cola.

Acción: Inicie AMQSCLM.

Explicación potencial: La cola tiene **CLWLUSEQ** = LOCAL o **CLWLUSEQ** = QMGR y el argumento **-u** no está definido.

Acción: Establezca el parámetro **-u** o cambie la configuración de la cola o del gestor de colas a ANY.

Explicación potencial: No hay instancias activas de la cola en el clúster.

Acción: Compruebe si hay instancias de la cola con un valor de 1 o superior en **CLWLPRTY**.

Posible explicación: las instancias de la cola remota tienen consumidores (**IPPROCS** >=1) pero están inactivos en esos gestores de colas (**CLWLPRTY** =0) porque AMQSCLM no está supervisando estas instancias remotas.

Acción: Asegúrese de que AMQSCLM esté ejecutando en esos gestores de colas y/o que la cola esté en la lista de colas que se supervisan comprobando los valores de los parámetros **-q** y **-f**.

Explicación de potencial: Las instancias de cola remota están activas (**CLWLPRTY** =1), pero aparecen como inactivas en el gestor de colas local (**CLWLPRTY** =0). Esta situación se debe a que el valor **CLWLPRTY** actualizado no se está propagando a este gestor de colas.

Acción: Asegúrese de que los gestores de colas remotos estén conectados al menos con uno de los gestores de colas de repositorio completo del clúster. Asegúrese de que los gestores de colas de repositorio completo funcionan correctamente. Compruebe que están ejecutando los canales entre los gestores de colas de repositorio completo y los gestores de colas supervisados.

Explicación potencial: Los mensajes no se confirman y, por tanto, no son recuperables.

Acción: Compruebe que la aplicación emisora está funcionando correctamente.

Explicación potencial: AMQSCLM no tiene acceso a la cola local donde se encolan los mensajes.

Acción: Compruebe si AMQSCLM está ejecutando como un usuario con autorización suficiente para acceder a la cola.

Explicación de potencial: El servidor de comandos del gestor de colas no está ejecutando.

Acción: Arranque el servidor de comandos en el gestor de colas.

Explicación de potencial: Se ha producido un error en AMQSCLM.

Acción: Compruebe si hay errores en los archivos de informe.

Explicación potencial: Las instancias de cola remota están activas (CLWLPRTY=1), pero los mensajes no se pueden transferir a esas instancias de cola porque el canal emisor de clúster del gestor de colas local no está ejecutando. Esto suele ir acompañado de una advertencia CLM0030W en el registro de informe amqsclm.

Acción: Compruebe el estado de los canales emisores de clúster del gestor de colas local al gestor (o gestores) de colas remotos con una instancia activa de la cola.

ALW *Programa de ejemplo para Connection Endpoint Lookup (CEPL)*

El ejemplo de IBM MQ Connection Endpoint Lookup proporciona un módulo de salida sencillo, pero potente, que ofrece a los usuarios de IBM MQ una forma de recuperar definiciones de conexión de un repositorio LDAP como, por ejemplo, Tivoli Directory Server.

El cliente de Tivoli Directory Server v6.3 tiene que estar instalado para que se pueda utilizar CEPL.

Para usar este ejemplo, se requieren conocimientos prácticos de administración de IBM MQ en las plataformas soportadas.

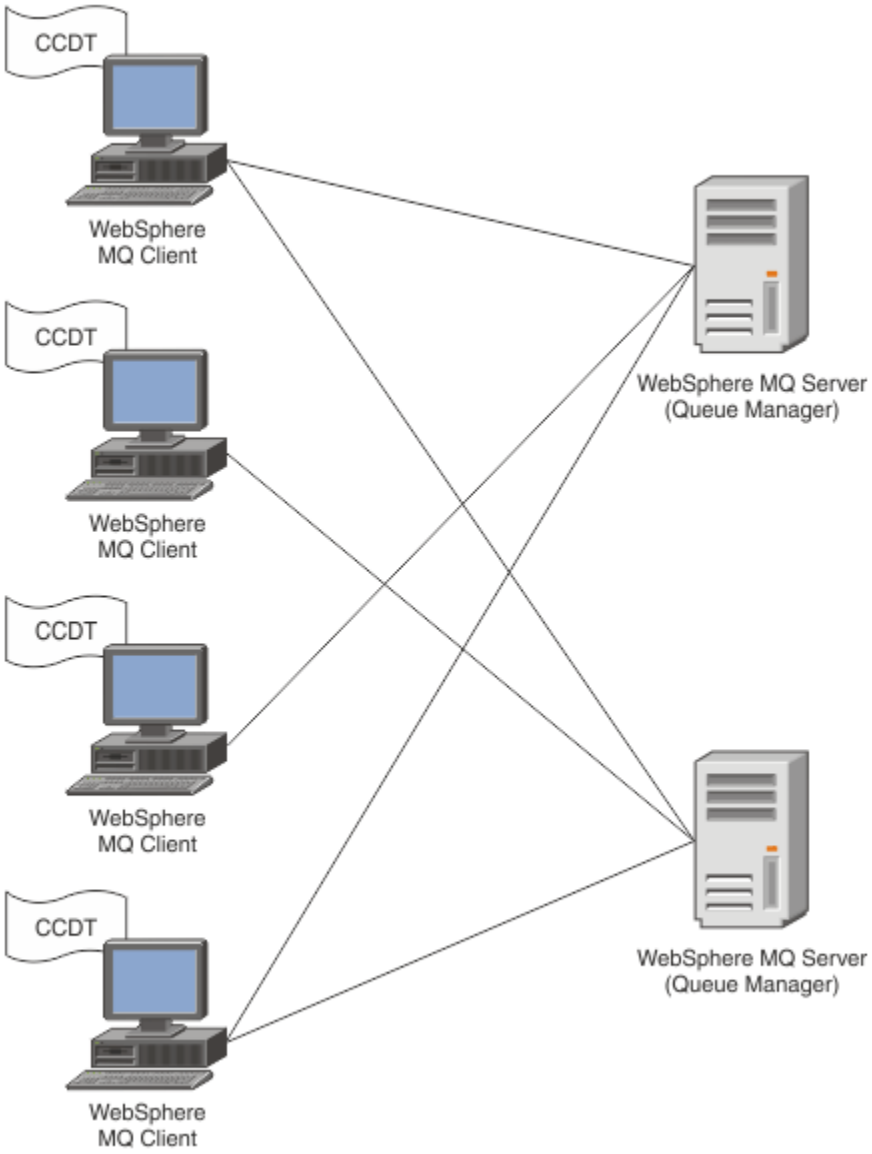
Windows Linux AIX *Introducción*

Configure un repositorio global, por ejemplo, un directorio LDAP (Lightweight Directory Access Protocol), para almacenar definiciones de conexión de cliente como ayuda para el mantenimiento y la administración.

Utilización de una aplicación cliente de IBM MQ para establecer una conexión con un gestor de colas a través de una tabla de definición de conexión de cliente (CCDT).

La CCDT se crea a través de la interfaz estándar de administración MQSC de IBM MQ. El usuario debe estar conectado a un gestor de colas para poder crear definiciones de conexión de cliente, aunque los datos contenidos en la definición no estén restringidos al gestor de colas. El archivo

CCDT generado se debe distribuir manualmente entre las máquinas cliente y las aplicaciones.

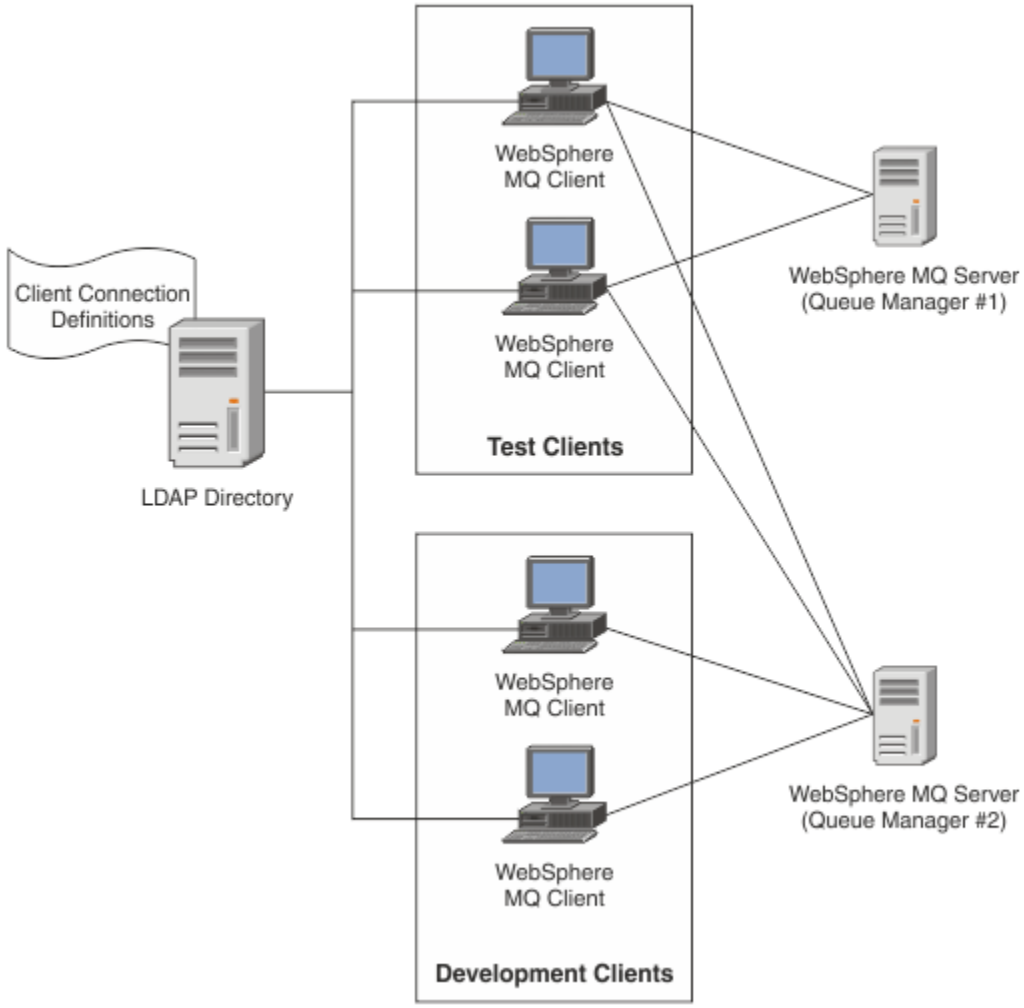


El archivo CCDT se debe distribuir a cada cliente de IBM MQ. En los casos en que haya miles de clientes en local o de forma global, pronto será difícil de mantener y administrar. Se necesita un enfoque más flexible para garantizar que cada cliente tenga a su disposición las definiciones de cliente correctas.

Un enfoque de este tipo es almacenar las definiciones de conexión de cliente en un repositorio global como, por ejemplo, un directorio LDAP (Lightweight Directory Access Protocol). Un directorio LDAP también puede proporcionar servicios de seguridad, indexación y búsqueda adicionales, permitiendo así a cada cliente acceder sólo a las definiciones de conexión que les pertenecen.

El directorio LDAP se puede configurar de modo que sólo estén disponibles definiciones específicas para determinados grupos de usuarios. Por ejemplo, los clientes de prueba pueden acceder tanto al gestor de

colas #1 como a #2, mientras que los clientes de desarrollo solo pueden acceder al gestor de colas #2 .



El módulo de salida puede buscar un repositorio LDAP, por ejemplo, IBM Tivoli Directory Server, para recuperar las definiciones de canal. Utilizando estas definiciones de conexión, una aplicación cliente de IBM MQ puede establecer la conexión con un gestor de colas.

El módulo de salida es un módulo de salida de preconexión que habilita la obtención de la definición de canal durante la llamada MQCONN/MQCONNX desde un repositorio LDAP.

El módulo de salida y el esquema se pueden implementar mediante:

- clientes que ya han creado una base de conocimientos utilizando la tecnología basada en el archivo CCDT existente y desean facilitar la administración y los costes de distribución.
- clientes existentes que ya emplean su propia tecnología propietaria para distribuir definiciones de conexión de cliente.
- clientes nuevos o existentes que actualmente no utilizan ningún tipo de solución de conexión de cliente y desean utilizar las características que ofrece IBM MQ.
- clientes nuevos o existentes que desean utilizar o ajustar directamente su modelo de mensajería en línea con cualquier arquitectura de negocio de LDAP actual.

ALW *Entornos soportados*

Verifique que tiene un sistema operativo soportado y el software relevante antes de ejecutar el ejemplo de búsqueda de punto final de conexión.

El programa de ejemplo de búsqueda de puntos finales de conexión de IBM MQ requiere el software siguiente:

- IBM WebSphere MQ 7.0 o posterior.
- Tivoli Directory Server V6.3 Client o posterior.

Sistemas operativos soportados:

1. Windows (7/8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 y v5 en System p
   - SUSE v9 y v10 en System p
   - RHEL v4 y v5 x86-64 32 bits y 64 bits
   - SUSE v9 y v10 x86-64 32 bits y 64 bits

Nota: El ejemplo no está disponible para las plataformas siguientes:

- z/OS
- IBM i

ALW *Instalación y configuración*
Instalación y configuración del módulo de salida y el esquema de punto final de conexión.

Instalación del módulo de salida

Durante la instalación de IBM MQ, el módulo de salida se instala en `tools/samples/c/preconnexit/bin`. Para plataformas de 32 bits, el módulo de salida se debe copiar en `exit/`*`installation_name`*`/` antes de que se pueda utilizar. Para las plataformas de 64 bits, el módulo de salida debe copiarse en exit64/*nombre_instalación*/ para poder utilizarse.

Instalación del esquema de punto final de conexión

La salida utiliza el esquema de punto final de conexión, `ibm-amq.schema`. El archivo de esquema debe importarse a cualquier servidor LDAP para poder utilizar la salida. Después de importar el esquema, se deben añadir los valores de los atributos.

A continuación, se muestra un ejemplo para importar el esquema de punto final de conexión. En el ejemplo, se supone que se utiliza IBM Tivoli Directory Server (ITDS).

- Asegúrese de que IBM Tivoli Directory Server se esté ejecutando, y copie o envíe por FTP el archivo `ibm-amq.schema` al servidor ITDS.
- En el servidor ITDS, especifique el mandato siguiente para instalar el esquema en el almacén ITDS, donde *ID de LDAP* y *contraseña de LDAP* son el DN raíz y la contraseña del servidor LDAP:

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- En una ventana de mandatos, especifique el siguiente mandato o utilice una herramienta de terceros para examinar el esquema para verificarlo:

```
ldapsearch objectclass=ibm-amqClientConnection
```

Consulte la documentación del servidor LDAP para obtener más detalles sobre la importación del archivo de esquema.

Configuración

Se debe añadir una nueva sección denominada PreConnect al archivo de configuración del cliente, por ejemplo `mqclient.ini`. La sección PreConnect contiene las siguientes palabras clave:

Módulo
El nombre del módulo que contiene el código de salida de la API. Si este campo contiene la vía de acceso completa del módulo, se utiliza tal cual. De lo contrario, se buscará en la carpeta `exit` o `exit64` en la instalación de IBM MQ.

Función
El nombre del punto de entrada funcional en la biblioteca que contiene el código de salida `LdapPreConnect` . La definición de función se ajusta al prototipo de función de la empresa.

Atención: Debe eliminar las comillas en la sentencia de función cuando especifique el punto de entrada de salida real.

Datos
URI del repositorio LDAP que contiene definiciones de canal.

El siguiente fragmento de código es un ejemplo de los cambios necesarios en el archivo `mqclient.ini`.

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Descripción general de una salida y un esquema*
Sintaxis y parámetros utilizados para establecer una conexión con un gestor de colas.

IBM MQ 9.3 define la sintaxis siguiente para un punto de entrada en un módulo de salida.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Durante la ejecución de la llamada MQCONN/X, el cliente C de IBM MQ carga el módulo de salida que contiene una implementación de la sintaxis de la función. Luego invoca una función de salida para recuperar las definiciones de canal. Las definiciones de canal recuperadas se utilizan a continuación para establecer conexión con un gestor de colas.

Parámetros

pExitParms
Tipo: PMQNXP entrada/salida

Estructura del parámetro de salida PreConnection. El invocador de la salida asigna y mantiene dicha estructura.

```
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
Tipo: PMQCHAR entrada/salida

Nombre del gestor de colas. En la entrada, este parámetro es la serie de filtro suministrada a la llamada de la API MQCONN a través del parámetro **QMgrName**. Este campo se puede estar en blanco, ser explícito o contener determinados caracteres de comodín. El campo lo modifica la salida. El parámetro es NULL cuando se invoca la salida con MQXR_TERM.

ppConnectOpts

Tipo: ppConnectOpts entrada/salida

Opciones que controlan la acción de MQCONNX. Este es un puntero a una estructura de opciones de conexión MQCNO que controla la acción de la llamada a la API MQCONN. El parámetro es NULL cuando se invoca la salida con MQXR_TERM. El cliente MQI siempre proporciona una estructura MQCNO a la salida, aunque la aplicación no la haya proporcionado originalmente. Si una aplicación proporciona una estructura MQCNO, el cliente realiza un duplicado para transferirla a la salida donde se modifica. El cliente conserva la propiedad de MQCNO. Un MQCD al que se hace referencia en MQCNO tiene prioridad sobre cualquier definición de conexión proporcionada mediante la matriz. El cliente utiliza la estructura MQCNO para conectarse con el gestor de colas y los demás se ignoran.

pCompCode

Tipo: PMQLONG entrada/salida

Código de terminación. Puntero a un MQLONG que recibe el código de terminación de salidas. Tiene que ser uno de los valores siguientes:

- MQCC_OK - Terminación satisfactoria
- MQCC_WARNING -Aviso (terminación parcial)
- MQCC_FAILED -La llamada ha fallado

pReason

Tipo: PMQLONG entrada/salida

Razón que complementa a pCompCode. Puntero a un MQLONG que recibe el código de razón de salida. Si el código de terminación es MQCC_OK, el único valor válido es: MQRC_NONE - (0, x '000') No hay razón por informar.

Si el código de terminación es MQCC_FAILED o MQCC_WARNING, la función de salida puede establecer el campo de código de razón a cualquier valor de MQRC_* válido.

ALW *Información de contexto LDAP de MQ*

La salida utiliza la siguiente estructura de datos para la información de contexto.

MQNLDAPCTX

La estructura MQNLDAPCTX tiene el prototipo C siguiente.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX *Código de ejemplo para crear una salida de búsqueda de punto final de conexión*

Puede utilizar los fragmentos de código de ejemplo para compilar el origen en AIX, o Linux, o Windows.

Compilación del código fuente

Puede compilar el código fuente con cualquier biblioteca de cliente LDAP, por ejemplo, las bibliotecas de cliente de IBM Tivoli Directory Server V6.3. En esta documentación se supone que están utilizando las bibliotecas cliente de Tivoli Directory Server V6.3.

Nota: La biblioteca de salida de conexión previa está soportada en los siguientes servidores LDAP:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Los fragmentos de código siguientes describen cómo compilar las salidas:

Windows **Compilación de la salida en la plataforma Windows**

Se puede utilizar el siguiente fragmento de código para compilar el fuente de la salida:

```
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
```

Nota: Si se están usando las bibliotecas cliente de IBM Tivoli Directory Server V6.3 compiladas con el compilador de Microsoft Visual Studio 2003, puede que se obtengan avisos al compilar dichas bibliotecas de IBM Tivoli Directory Server V6.3 con el compilador de Microsoft Visual Studio 2012 o posterior.

Linux AIX **Compilación de la salida en AIX, o Linux**

El fragmento de código siguiente es para compilar el código fuente de salida en Linux. Algunas opciones de compilador pueden diferir en AIX.

```
#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 entrega bibliotecas de enlace estáticas y, también, dinámicas, pero solo puede utilizar un tipo de biblioteca. En este script se asume el uso de bibliotecas estáticas.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *Invocación del módulo de salida de PreConnect*

El módulo de salida de PreConnect se puede invocar con tres códigos de razón distintos: el código de razón MQXR_INIT para inicializar y establecer una conexión con un servidor LDAP, el código de razón MQXR_PRECONNECT para recuperar definiciones de canal de un servidor LDAP o el código de razón MQXR_TERM cuando la salida se va a limpiar.

MQXR_INIT

La salida se invoca con el código de razón MQXR_INIT para la inicialización y establecimiento de una conexión a un servidor LDAP.

Antes de la llamada MQXR_INIT, el campo pExitDataPtr de la estructura MQNXP se llena con el atributo Datos de la stanza PreConnect dentro del archivo `mqclient.ini` (es decir, el LDAP).

Un URL de LDAP consiste en, como mínimo, el protocolo, el nombre de host, el número de puerto y el DN base para la búsqueda. La salida analiza el URL de LDAP contenido en el campo pExitDataPtr, asigna una estructura de contexto de búsqueda de LDAP MQNLDAPCTX y la llena en consecuencia. La dirección de esta estructura se almacena en el campo pExitUserAreaPtr. Si no se analizan correctamente el URL de LDAP, se producirá el error MQCC_FAILED.

En este punto, la salida se conecta y se enlaza con el servidor LDAP utilizando los parámetros **MQNLDAPCTX**. Los manejadores de API de LDAP resultantes también se almacenan dentro de esta estructura.

MQXR_PRECONNECT

El módulo de salida se invoca con el código de razón MQXR_PRECONNECT para recuperar definiciones de canal de un servidor LDAP.

La salida busca en el servidor LDAP definiciones de canales que coincidan con el filtro determinado. Si el **QMgrNameparameter** contiene un nombre de gestor de colas específico, la búsqueda devuelve todas las definiciones de canal para las que el valor de atributo LDAP **ibm-amqQueueManagerName** coincida con el nombre del gestor de colas especificado.

Si el parámetro **QMgrName** es '*' o ' ' (espacio), la búsqueda devuelve todas las definiciones de canal para las que el atributo de punto final **ibm-amqIsClientDefault Connection** está establecido en TRUE.

Después de una búsqueda satisfactoria, la salida prepara una definición (o una matriz de definiciones) de MQCD y vuelve al emisor de la llamada.

MQXR_TERM

La salida se invoca con este código de razón cuando se va a limpiar la salida. Durante esta limpieza, la salida se desconecta del servidor LDAP y libera toda la memoria asignada y mantenida por la salida, incluida la estructura MQNLDAPCTX, la matriz de puntero y cada MQCD a la que hace referencia. Cualquier otro campo se establece en los valores predeterminados. Los parámetros de salida **pQMgrName** y **ppConnectOpts** no se utilizan durante una salida con el código de razón MQXR_TERM y podrían ser NULL.

Referencia relacionada

Stanza PreConnect del archivo de configuración del cliente

ALW *Esquemas de LDAP*

Los datos de conexión cliente se almacenan en un repositorio global llamado directorio LDAP (Lightweight Directory Access Protocol, Protocolo ligero de acceso a directorios). Un cliente IBM MQ utiliza un directorio LDAP para obtener las definiciones de conexión. La estructura de las definiciones de conexión de cliente de IBM MQ en el directorio LDAP se conoce como esquema LDAP. Un esquema LDAP es la colección de definiciones de tipo de atributo, definiciones de clase de objeto y otra información que un servidor utiliza para determinar si un filtro o una aserción de valor de atributo coincide con los atributos de una entrada, y si hay que permitir, añadir o modificar operaciones.

Almacenamiento de datos en el directorio LDAP

Las definiciones de conexión cliente se encuentran bajo una rama específica del árbol de directorios conocida como punto de conexión. Al igual que todos los demás nodos de un directorio LDAP, el punto

de conexión tiene un nombre distinguido (DN) asociado. Puede utilizar este nodo como punto de partida de cualquier consulta que realice en el directorio. Utilice el filtrado al consultar el directorio LDAP para devolver un subconjunto de definiciones de conexión cliente. Puede restringir el acceso a los subárboles en función de los permisos otorgados en otras partes del árbol de directorios; por ejemplo, a usuarios, departamentos o grupos.

Definición de atributos y clases propios

Almacene la definición del canal cliente modificando el esquema LDAP. Todas las definiciones de datos de LDAP requieren objetos y atributos. Los objetos y los atributos se identifican mediante un número de identificador de objeto (OID), que identifica de forma exclusiva el objeto o el atributo. Todas las clases dentro de un esquema LDAP heredan directa o indirectamente del objeto superior. El objeto de definición de canal cliente contiene los atributos del objeto superior. Todas las definiciones de datos de LDAP requieren objetos y atributos:

- Las definiciones de objeto son colecciones de atributos LDAP.
- Los atributos son tipos de datos LDAP.

La descripción de cada atributo y cómo se correlacionan con las propiedades normales de IBM MQ se describen en Atributos LDAP.

ALW *atributos LDAP*

Los atributos LDAP definidos son específicos para IBM MQ y se correlacionan directamente con las propiedades de conexión de cliente.

Atributos de serie de directorio de canal de cliente de IBM MQ

Los atributos de serie de caracteres con su correlación con las propiedades de IBM MQ se listan en la tabla siguiente. Los atributos pueden contener valores de sintaxis directoryString (Unicode codificado en UTF-8, es decir, un sistema de codificación de byte variable que incluye IA5/ASCII como un subconjunto). La sintaxis se especifica mediante el número de identificación de objeto (OID).

Tabla 166. Atributos de serie de directorio de canal de cliente de IBM MQ

Atributo LDAP	Descripción	Propiedad de IBM MQ
CN	El nombre común formado por el nombre de canal y el nombre del gestor de colas de definición.	
ibm-amqChannelName	El nombre de la definición de canal.	CHANNEL
ibm-amqConnectionName	El identificador de la conexión de comunicación.	CONNAME
ibm-amqDescription	La descripción del canal.	DESCR
ibm-amqLocalAddress	La dirección de comunicación local del canal.	LOCLADDR
ibm-amqModeName	Nombre de la modalidad LU 6.2.	MODENAME
ibm-amqPassword	La contraseña que se puede utilizar.	PASSWORD
ibm-amqQueueManagerName	El nombre del gestor de colas o del grupo de gestores de colas al que una aplicación cliente de IBM MQ puede solicitar la conexión.	QMNAME
ibm-amqSecurityExitUserData	Los datos de usuario que se pasan a la salida de seguridad.	SCYDATA
ibm-amqSecurityExitName	El nombre del programa de salida que va a ejecutar la salida de seguridad del canal.	SCYEXIT
ibm-amqSslCipherSpec	Una CipherSpec única para una conexión TLS.	SSLCIPH

Tabla 166. Atributos de serie de directorio de canal de cliente de IBM MQ (continuación)		
Atributo LDAP	**Descripción**	**Propiedad de IBM MQ**
ibm-amqSslPeerName	Comprueba el nombre distinguido (DN) del certificado del gestor de colas o el cliente de igual en el otro extremo de un canal de IBM MQ.	SSLPEER
ibm-amqTransactionProgramName	El nombre del programa de transacción.	TPNAME
ibm-amqUserID	El ID de usuario que debe utilizar el MCA al intentar iniciar una sesión SNA segura con un MCA remoto.	USERID

Atributos de enteros de conexión de cliente de IBM MQ

Los atributos con valores predefinidos (por ejemplo, un tipo enumerado) se almacenan como enteros estándar. Estos valores se almacenan en el directorio LDAP como valores enteros, y no utilizando el nombre de constante asociado.

Tabla 167. Atributos de enteros de directorio de canal de cliente de IBM MQ		
Atributo LDAP	**Descripción**	**Propiedad de IBM MQ**
ibm-amqConnectionAffinity	Determina si las aplicaciones cliente, que se conectan varias veces a través del mismo nombre de gestor de colas, utilizan el mismo canal de cliente.	AFINIDAD
ibm-amqClientChannelWeight	Una ponderación para influir sobre qué definición de canal de conexión de cliente se utiliza.	CLNTWGHT
ibm-amqHeartBeatInterval	El tiempo aproximado entre flujos de latidos que se van a pasar de un MCA de envío cuando no hay mensajes en la cola de transmisión.	HBINT
ibm-amqKeepAliveInterval	Valor de tiempo de espera para un canal.	KAINT
ibm-amqMaximumMessageLength	La longitud máxima de un mensaje que se puede transmitir en un canal.	MAXMSGL
ibm-amqSharingConversations	El número máximo de conversaciones que comparte cada instancia del canal TCP/IP.	SHARECNV
ibm-amqTransportType	El tipo de transporte va a utilizar.	TRPTYPE

Atributo booleano de canal de cliente de IBM MQ

Este atributo booleano no está correlacionado con ninguna propiedad de IBM MQ. La sintaxis de este atributo indica un valor booleano.

Tabla 168. Atributo booleano de canal de cliente de IBM MQ	
Atributo LDAP	**Descripción**
ibm-amqIsClientDefault	Este atributo booleano se define para resolver el problema de buscar las entradas cuyo atributo ibm-amqQueueManagerName no se haya definido.

Atributos de la lista de canales de cliente de IBM MQ

Las propiedades de IBM MQ se almacenan como un atributo de lista separada por comas en el directorio LDAP. Los atributos se definen de la misma forma que los otros atributos de serie de

directorio. Los atributos de lista junto con su correlación con las propiedades de IBM MQ se describen en la tabla siguiente.

Tabla 169. Atributos de la lista de canales de cliente de IBM MQ

Atributo LDAP	Descripción	Propiedad de IBM MQ
ibm-amqHeaderCompression	Una lista de las técnicas de compresión de datos de cabecera admitidas por el canal.	COMPHDR
ibm-amqMessageCompression	Una lista de las técnicas de compresión de datos de mensaje admitidas por el canal.	COMPMSG
ibm-amqSendExitUserData	Los datos de usuario que se pasan a la salida de envío.	SENDDATA
ibm-amqSendExitUserName	El nombre del programa de salida que va a ejecutar la salida de envío del canal.	SENDEXIT
ibm-amqReceiveExitUserData	Los datos de usuario que se pasan a la salida de recepción.	RCVDATA
ibm-amqReceiveExitName	El nombre del programa de salida de usuario que va a ejecutar la salida de usuario de recepción de canal.	RCVEXIT

ALW *Nombre común*

El nombre común (CN) está formado por el nombre de canal y el nombre del gestor de colas de definición.

Es un atributo preexistente.

El formato del CN es:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

Por ejemplo:

```
CN=TC1(QM_T1)
```

Solo se puede especificar un valor en este atributo.

Este atributo es un atributo de serie y los valores no distinguen entre mayúsculas y minúsculas. La coincidencia de subserie se ignora. La coincidencia de subserie es una regla de coincidencia utilizada en el subesquema que especifica el comportamiento del atributo en un filtro de búsqueda, utilizando una subserie (por ejemplo, CN=jim *, donde CN es un atributo) y contiene uno o más comodines.

ALW *ibm-amqChannelName*

Este atributo especifica el nombre de una definición de canal.

Este atributo tiene un valor de cadena única con un máximo de 20 caracteres donde no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda, usando una subcadena, y contiene uno o más comodines.

ALW *ibm-amqDescription*

Este atributo de LDAP proporciona la descripción de canal.

Este atributo tiene un valor de cadena única con un máximo de 64 bytes en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqConnectionName*
Este atributo de LDAP es el identificador de conexión de comunicaciones. Especifica los enlaces de comunicaciones concretos que tiene que utilizar un canal.

Este atributo tiene un valor de cadena única con un máximo de 264 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqLocalAddress*
Este atributo especifica la dirección de comunicaciones local de un canal.

Este atributo tiene un valor de cadena única con un máximo de 48 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqModeName*
Este atributo se utiliza en las conexiones LU 6.2. Proporciona una definición adicional a las características de sesión de una conexión cuando se realiza una asignación de sesión de comunicación.

Este atributo tiene un único valor de cadena de exactamente 8 caracteres, donde no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqPassword*
Este atributo de LDAP especifica la contraseña que el MCA puede utilizar cuando intenta iniciar una sesión de LU 6.2 segura con un MCA remoto.

Este atributo tiene un valor entero simple con un máximo de 12 dígitos. No se trata de un atributo preexistente.

ALW *ibm-amqQueueManagerName*
Este atributo especifica el nombre del gestor de colas o el grupo de gestores de colas a los que una aplicación cliente de IBM MQ puede solicitar una conexión.

Este atributo tiene un valor de cadena única con un máximo de 48 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

Referencia relacionada
“ibm-amqIsClientDefault” en la página 1187
Este atributo booleano soluciona el problema de buscar entradas en las que el atributo ibm-amqQueueManagerName no se ha definido.

ALW *ibm-amqSecurityExitUserData*
Este atributo de LDAP especifica los datos de usuario que se pasan a una salida de seguridad.

Este atributo tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqSecurityExitName*
Este atributo de LDAP especifica el nombre del programa de salida que tiene que ser ejecutado por la salida de seguridad de canal.

Déjelo en blanco si no hay ninguna salida de seguridad de canal en vigor.

Este atributo tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. Este atributo no es de salida previa.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqSslCipherSpec*
Este atributo de LDAP especifica una CipherSpec única en una conexión TLS.

Este atributo tiene un valor de cadena única con un máximo de 32 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqSslPeerName*
Este atributo LDAP se utiliza para comprobar el nombre distinguido (DN) del certificado del gestor de colas o el cliente homólogos en el otro extremo de un canal IBM MQ.

Este atributo LDAP tiene un valor de cadena única con un máximo de 1024 bytes en la que no se distingue entre mayúsculas y minúsculas. No es preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqTransactionProgramName*
Este atributo de LDAP especifica el nombre del programa de transacción. Se utiliza en las conexiones LU 6.2.

Este atributo tiene un valor de cadena única con un máximo de 64 caracteres en la que no se distingue entre mayúsculas y minúsculas. No es preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqUserID*
Este atributo de LDAP especifica el ID de usuario que tiene uqe usar el MCA al intentar iniciar una sesión SNA segura con un MCA remoto.

Este atributo tiene un único valor de cadena de exactamente 12 caracteres, donde no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

ALW *ibm-amqConnectionAffinity*
Este atributo de LDAP especifica si las aplicaciones cliente que se conectan múltiples veces usando el mismo nombre de gestor de colas usan el mismo canal cliente.

Este atributo tiene un único valor entero. No se trata de un atributo preexistente.

ALW *ibm-amqClientChannelWeight*
Este atributo de LDAP especifica una ponderación que influye en qué definición de canal de conexión de cliente se utiliza.

El atributo de ponderación de canal de cliente se utiliza para decantar la selección de definiciones de canal de cliente cuando se dispone de más de una definición adecuada.

Este atributo tiene un único valor entero. No se trata de un atributo preexistente.

ALW *ibm-amqHeartBeatInterval*
Este atributo de LDAP especifica el tiempo aproximado entre los flujos de latidos que se pasan desde un MCA de envío cuando no hay mensajes en la cola de transmisión.

Este atributo tiene un único valor entero. No se trata de un atributo preexistente. El valor predeterminado es 1. El valor predeterminado se establece en la operación de variable de entorno MQSERVER actual.

ALW *ibm-amqKeepAliveInterval*
Este atributo de LDAP se utiliza para especificar un valor de tiempo de espera en un canal.

El valor de este atributo se pasa a la pila de comunicaciones y especifica la temporización de estado activo (keepalive) del canal. Se puede usar para especificar un valor de estado activo diferente para cada canal.

Este atributo tiene un único valor entero. No se trata de un atributo preexistente.

ALW *ibm-amqMaximumMessageLength*
Este atributo de LDAP especifica la longitud máxima de un mensaje que se puede transmitir en un canal.

El valor predeterminado de este atributo es 104857600 conforme a la operación de la variable de entorno MQSERVER actual. Este atributo tiene un único valor entero y no es preexistente.

ALW *ibm-amqSharingConversations*
Este atributo de LDAP especifica el número máximo de conversaciones que comparten cada instancia de canal TCP/IP.

Este atributo tiene un único valor entero. Este atributo no es preexistente.

ALW *ibm-amqTransportType*
Este atributo de LDAP especifica el tipo de transporte que se va a utilizar.

Este atributo tiene un único valor entero. No se trata de un atributo preexistente.

ALW *ibm-amqIsClientDefault*
Este atributo booleano soluciona el problema de buscar entradas en las que el atributo ibm-amqQueueManagerName no se ha definido.

Los módulos de salida de preconexión suelen buscar en servidores LDAP usando el valor del atributo ibm-amqQueueManagerName como criterio de búsqueda. Una consulta de este tipo devolvería todas las entradas en las que el valor del atributo ibm-amqQueueManagerName coincide con el nombre del gestor de colas especificado en la llamada MQCONN/X. Sin embargo, al utilizar las tablas de definición de canal de cliente (CCDT), puede establecer el nombre del gestor de colas en una llamada MQCONN/X en blanco o añadir un asterisco (*) como prefijo al nombre. Si el nombre del gestor de colas está en blanco, el cliente se conecta al gestor de colas predeterminado. Si el nombre está prefijado con un asterisco (*) en el gestor de colas, el cliente se conecta con cualquier gestor de colas.

Del mismo modo, el atributo ibm-amqQueueManagerName de una entrada se puede dejar sin definir. En tal caso, se espera que el cliente que utiliza esta información de punto final se pueda conectar con cualquier gestor de colas. Por ejemplo, una entrada contiene las líneas siguientes:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

En este ejemplo, el cliente intenta conectarse al gestor de colas especificado que se ejecuta en `myhost`.

Sin embargo, en los servidores LDAP, no se efectúan búsquedas de valores de atributo no definidos. Por ejemplo, si una entrada contiene la información de conexión excepto ibm-amqQueueManagerName, el resultado de la búsqueda no incluirán esta entrada. Para solucionar este problema, se puede definir ibm-amqIsClientDefault. Se trata de un atributo booleano y se supone que tiene un valor de FALSE si no se configura.

En el caso de las entradas en las que no se ha definido ibm-amqQueueManagerName y de las que se espera que formen parte de la búsqueda, establezca ibm-amqIsClientDefault a TRUE. Cuando se especifica un espacio en blanco o un asterisco (*) como nombre del gestor de colas en una llamada a MQCONN/X, la salida de preconexión busca en el servidor LDAP todas las entradas donde el valor de atributo ibm-amqIsClientDefault se haya establecido a TRUE.

Nota: No establezca ni defina el atributo ibm-amqQueueManagerName si ibm-amqIsClientDefault está establecido a TRUE.

Referencia relacionada

“ibm-amqQueueManagerName” en la página 1185
Este atributo especifica el nombre del gestor de colas o el grupo de gestores de colas a los que una aplicación cliente de IBM MQ puede solicitar una conexión.

ALW *ibm-amqHeaderCompression*
Este atributo de LDAP es una lista de las técnicas de compresión de datos de cabecera soportadas por el canal.

El tamaño máximo de este atributo es de 48 caracteres. No se trata de un atributo preexistente.

Solo se puede especificar un valor en este atributo.

Este atributo de lista se especifica como cadenas de directorio en formato de separación por comas. Por ejemplo, el valor especificado para **ibm-amqHeaderCompression** es `0`, que se correlaciona con `NONE`. El cliente ignora los valores que rebasa el límite máximo permitido. Por ejemplo, ibm-amqHeaderCompression contiene un máximo de 2 enteros en la lista.

ALW *ibm-amqMessageCompression*
Este atributo de LDAP es una lista de las técnicas de compresión de datos de mensaje soportadas por el canal.

El tamaño máximo de este atributo es de 48 caracteres. No se trata de un atributo preexistente.

Este atributo no soporta valores múltiples.

Este atributo de lista se especifica como cadenas de directorio en formato de separación por comas. Por ejemplo, el valor especificado para este atributo es 1,2,4, que se correlaciona con la secuencia de compresión subyacente RLE, ZLIBFAST y ZLIBHIGH.

El cliente ignora los valores que rebasa el límite máximo permitido. Por ejemplo, ibm-amqMessageCompression contiene un máximo de 16 enteros en la lista.

ALW *ibm-amqSendExitUserData*
Este atributo de LDAP especifica los datos de usuario que se pasan a una salida de envío.

Este atributo LDAP tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

Nota: ibm-amqSendExitName y **ibm-amqSendExitUserData** tienen que ir sincronizados en parejas. Los datos de usuario tienen que estar sincronizados con el nombre de salida. Por tanto, si se especifica uno, el otro también tiene que especificarse simétricamente, aunque no contenga datos.

ALW *ibm-amqSendExitName*

Este atributo de LDAP especifica el nombre del programa de salida que tiene que ser ejecutado por la salida de envío de canal.

Este atributo tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

Nota: ibm-amqSendExitName y **ibm-amqSendExitUserData** tienen que ir sincronizados en parejas. Los datos de usuario tienen que estar sincronizados con el nombre de salida. Por tanto, si se especifica uno, el otro también tiene que especificarse simétricamente, aunque no contenga datos.

ALW *ibm-amqReceiveExitUserData*

Este atributo de LDAP especifica los datos de usuario que se pasan a una salida de recepción.

Se puede ejecutar una secuencia de salidas de recepción. La cadena de datos de usuario de una serie de salidas está separada por comas, espacios o ambos.

Este atributo tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

Nota: ibm-amqReceiveExitName y **ibm-amqReceiveExitUserData** tienen que ir sincronizados en parejas. Los datos de usuario tienen que estar sincronizados con el nombre de salida. Por tanto, si se especifica uno, el otro también tiene que especificarse simétricamente, aunque no contenga datos.

ALW *ibm-amqReceiveExitName*

Este atributo de LDAP especifica el nombre del programa de salida de usuario que tiene que ser ejecutado por la salida de usuario de recepción de canal.

Este atributo es una lista de los nombres de los programas que se van a ejecutar en sucesión. Déjelo en blanco si no hay ninguna salida de usuario de recepción de canal en vigor.

Este atributo tiene un valor de cadena única con un máximo de 999 caracteres en la que no se distingue entre mayúsculas y minúsculas. No se trata de un atributo preexistente.

La coincidencia de subserie se ignora. La coincidencia de subcadena es una regla de coincidencia, usada en subesquemas, que especifica el comportamiento del atributo en un filtro de búsqueda.

Nota: ibm-amqReceiveExitName y **ibm-amqReceiveExitUserData** tienen que ir sincronizados en parejas. Los datos de usuario tienen que estar sincronizados con el nombre de salida. Por tanto, si se especifica uno, el otro también tendrá que especificarse simétricamente, aunque no contenga ningún dato.

z/OS Utilización de los programas de ejemplo para z/OS

Las aplicaciones de procedimientos de ejemplo que se entregan con IBM MQ for z/OS ilustran los usos típicos de la interfaz de cola de mensajes (MQI).

Acerca de esta tarea

IBM MQ for z/OS también proporciona salidas de conversión de datos de ejemplo, que se describen en “Escribir salidas de conversión de datos” en la página 1009.

Todas las aplicaciones de ejemplo se proporcionan en forma de código fuente; varias de ellas también se suministran en formato ejecutable. Los módulos fuente incluyen pseudocódigo que describe la lógica del programa.

Nota: Aunque algunas de las aplicaciones de ejemplo tienen interfaces basadas en panel básicas, no tienen como objetivo mostrar cómo diseñar el aspecto de las aplicaciones. Para obtener más información sobre cómo diseñar interfaces basadas en panel para terminales no programables, consulte *Acceso común de usuario: Guía básica de diseño de interfaces* (SC26-4583) y su adición (GG22-9508). Proporcionan directrices que ayudan a diseñar aplicaciones coherentes tanto internamente como entre ellas.

Procedimiento

- Utilice los enlaces siguientes para obtener más información sobre los programas de ejemplo:
  - “Funcionalidades ilustradas en las aplicaciones de ejemplo para z/OS” en la página 1190
  - “Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS” en la página 1197
  - “Preparación de aplicaciones de ejemplo para el entorno TSO en z/OS” en la página 1200
  - “Preparación de las aplicaciones de ejemplo para el entorno CICS en z/OS” en la página 1202
  - “Preparación de la aplicación de ejemplo para el entorno IMS en z/OS” en la página 1206
  - “Ejemplos de Put en z/OS” en la página 1207
  - “Ejemplos de obtención en z/OS” en la página 1209
  - “Ejemplo de examen en z/OS” en la página 1212
  - “Ejemplo Imprimir mensaje (Print Message) en z/OS” en la página 1214
  - “Ejemplo de atributos de cola en z/OS” en la página 1218
  - “Ejemplo de gestor de correo en z/OS” en la página 1219
  - “Ejemplo de comprobación de crédito en z/OS” en la página 1226
  - “Ejemplo del manejador de mensajes en z/OS” en la página 1238
  - “Ejemplo de colocación asíncrona en z/OS” en la página 1242
  - “El ejemplo de consumo asíncrono por lotes en z/OS” en la página 1243
  - “El ejemplo de consumo asíncrono y de publicación/suscripción de CICS en z/OS” en la página 1244
  - “Ejemplo de publicación/suscripción en z/OS” en la página 1247
  - “Ejemplo de establecimiento y consulta de las propiedades de un mensaje en z/OS” en la página 1249

Tareas relacionadas

“Utilización de los programas de ejemplo en Multiplataformas” en la página 1086
Estos programas de procedimiento de ejemplo se entregan con el producto. Los ejemplos se escriben en C y COBOL y muestran los usos típicos de la interfaz de colas de mensajes (MQI).

z/OS *Funcionalidades ilustradas en las aplicaciones de ejemplo para z/OS*

En esta sección se resume la funcionalidad de MQI que se ilustra en cada una de las aplicaciones de ejemplo, se muestran los lenguajes de programación en los que está escrito cada ejemplo y el entorno en el ejecuta.

z/OS *Ejemplos de transferencia en z/OS*

Los programas de transferencia de ejemplo muestran cómo colocar mensajes en una cola utilizando la llamada MQPUT.

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQPUT

- MQCLOSE
- MQDISC

El programa se entrega en COBOL y C, y se ejecuta en el entorno de procesos por lotes y CICS. Consulte Tabla 172 en la página 1198 para la aplicación por lotes y Tabla 179 en la página 1203 para la aplicación CICS.

z/OS *Ejemplos de obtención en z/OS*

Los ejemplos de obtención muestran cómo obtener mensajes de una cola utilizando la llamada MQGET.

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

El programa se entrega en COBOL y C, y se ejecuta en el entorno de procesos por lotes y CICS. Consulte Tabla 172 en la página 1198 para la aplicación por lotes y Tabla 179 en la página 1203 para la aplicación CICS.

z/OS *Ejemplo de examinar en z/OS*

El ejemplo de examinar muestra cómo utilizar la opción Examinar para buscar un mensaje, imprimirlo y, a continuación recorrer los mensajes de una cola.

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQGET para examinar mensajes
- MQCLOSE
- MQDISC

El programa se entrega en los lenguajes COBOL, ensamblador, PL/I y C. La aplicación se ejecuta en el entorno de proceso por lotes. Consulte Tabla 173 en la página 1198 para ver la aplicación por lotes.

z/OS *Ejemplo de Impresión de mensaje (Print message) en z/OS*

El ejemplo de impresión de mensaje (Print Message) muestra cómo eliminar un mensaje de una cola e imprimir los datos en el mensaje, junto con todos los campos de su descriptor de mensaje. Puede, opcionalmente, mostrar todas las propiedades de mensaje asociadas con cada mensaje.

Al eliminar los caracteres de comentario de dos líneas en el módulo de origen, puede cambiar el programa de forma que examine, en lugar de eliminar, los mensajes de una cola. Este programa puede ser útil para diagnosticar problemas con una aplicación que está poniendo mensajes en una cola.

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQGET para eliminar mensajes de una cola (con una opción para examinar)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

El programa se entrega en el lenguaje C. La aplicación se ejecuta en el entorno de proceso por lotes. Consulte Tabla 174 en la página 1199 para ver la aplicación por lotes.

z/OS *Ejemplo de atributos de cola en z/OS*

El ejemplo de atributos de cola muestra cómo consultar y establecer los valores de los atributos de objetos IBM MQ for z/OS.

La aplicación utiliza estas llamadas MQI:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

El programa se entrega en los lenguajes COBOL, ensamblador y C. La aplicación se ejecuta en el entorno CICS. Consulte Tabla 180 en la página 1203 para la aplicación CICS.

z/OS *Ejemplo de gestor de correo en z/OS*

Consideraciones a tener en cuenta cuando se utiliza el ejemplo de gestor de correo.

El ejemplo de gestor de correo demuestra estas técnicas:

- Utilizar colas alias
- Utilizar una cola modelo para crear una cola dinámica temporal
- Utilizar colas de respuestas
- Utilización de puntos de sincronización en los entornos de proceso por lotes y CICS
- Envío de mandatos a la cola de entrada de mandatos del sistema
- Probar los códigos de retorno
- Envío de mensajes a gestores de colas remotos, utilizando una definición local de una cola remota o colocando mensajes directamente en una cola con nombre en un gestor de colas remoto

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Se proporcionan tres versiones de la aplicación:

- Una aplicación CICS escrita en COBOL
- Una aplicación TSO escrita en COBOL.
- Una aplicación TSO escrita en C.

Las aplicaciones TSO utilizan el adaptador de proceso por lotes de IBM MQ for z/OS e incluyen algunos paneles de ISPF.

Consulte la Tabla 177 en la página 1200 para la aplicación TSO y la Tabla 181 en la página 1204 para la aplicación CICS.

z/OS *Ejemplo de comprobación de crédito en z/OS*

Esta información contiene puntos a tener en cuenta cuando se utiliza el ejemplo de comprobación de crédito.

El ejemplo de comprobación de crédito es una suite de programas que muestra estas técnicas:

- Desarrollar una aplicación que se ejecuta en más de un entorno
- Utilizar una cola modelo para crear una cola dinámica temporal
- Utilizar un identificador de correlación
- Establecer y pasar información de contexto
- Utilizar la prioridad y persistencia de mensajes
- Iniciar programas mediante desencadenamiento
- Utilizar colas de respuestas
- Utilizar colas alias
- Utilizar una cola de mensajes no entregados
- Utilizar una lista de nombres
- Probar los códigos de retorno

La aplicación utiliza estas llamadas MQI:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET para examinar y obtener mensajes, utilizando las opciones de espera y señal, y para obtener un mensaje específico
- MQINQ
- MQSET
- MQCLOSE

El ejemplo se puede ejecutar como una aplicación CICS autónoma. No obstante, para mostrar cómo se diseña una aplicación de puesta en cola de mensajes que utiliza los recursos que proporcionan los entornos CICS e IMS, también se proporciona un módulo como un programa de proceso de mensajes por lotes de IMS.

Los programas CICS se entregan en C y COBOL. El programa IMS único se entrega en C.

Consulte Tabla 182 en la página 1204 para la aplicación CICS y consulte Tabla 184 en la página 1206 para la aplicación IMS.

z/OS *Ejemplo del manejador de mensajes en z/OS*

El ejemplo del Manejador de mensajes permite examinar, reenviar y suprimir mensajes en una cola.

La aplicación utiliza estas llamadas MQI:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

El programa se entrega en los lenguajes de programación C y COBOL. La aplicación se ejecuta en TSO. Consulte la Tabla 178 en la página 1201 para ver la aplicación TSO.

z/OS *Ejemplos de salida de colas distribuidas en z/OS*

Una tabla de programas fuente de ejemplos de salida de colas distribuidas.

Los nombres de los programas fuente de los ejemplos de salida de colas distribuidas se muestran en la tabla siguiente:

Tabla 170. Fuente de los ejemplos de salida de colas distribuidas

Nombre del miembro	Para el lenguaje	Descripción	Suministrado en la biblioteca
CSQ4BAX0	Ensamblador	Programa fuente	SCSQASMS
CSQ4BCX1	C	Programa fuente	SCSQC37S
CSQ4BCX2	C	Programa fuente	SCSQC37S
CSQ4BCX4	C	Programa fuente	SCSQC37S

Nota: Los programas fuente se editan mediante enlace con CSQXSTUB.

z/OS *Ejemplos de salida de conversión de datos en z/OS*

Se proporciona un esqueleto para una rutina de salida de conversión de datos y se suministra un ejemplo con IBM MQ que ilustra la llamada MQXCNVC.

Los nombres de los programas de origen de los ejemplos de salida de conversión de datos se listan en la tabla siguiente:

Tabla 171. Origen de los ejemplos de salida de conversión de datos (solo lenguaje ensamblador)

Nombre del miembro	Descripción	Suministrado en la biblioteca
CSQ4BAX8	Programa fuente	SCSQASMS
CSQ4BAX9	Programa fuente	SCSQASMS
CSQ4CAX9	Programa fuente	SCSQASMS

Nota: Los programas fuente se editan mediante enlaces con CSQASTUB.

Consulte “Escribir salidas de conversión de datos” en la página 1009 para obtener más información.

z/OS *Ejemplos de publicación/suscripción en z/OS*

Los programas de ejemplo de publicación/suscripción muestran el uso de las características de publicación y suscripción en IBM MQ.

Hay cuatro programas de ejemplo de lenguaje de programación C y dos de lenguaje de programación COBOL que demuestran cómo programar la interfaz de publicación/suscripción de IBM MQ.

Las aplicaciones utilizan estas llamadas MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC

- MQCRTMH
- MQDLTMH
- MQINQMP

Los programas de ejemplo de publicación/suscripción se entregan en los lenguajes de programación C y COBOL. Las aplicaciones de ejemplo se ejecutan en el entorno de proceso por lotes. Consulte Ejemplos de publicación/suscripción para las aplicaciones de proceso por lotes.

z/OS *Configuración del gestor de colas para que acepte conexiones de cliente en z/OS*

Para poder ejecutar las aplicaciones de ejemplo, en primer lugar, debe crear un gestor de colas. A continuación, puede configurar el gestor de colas para que acepte de forma segura las solicitudes de conexión de entradas de las aplicaciones que se ejecutan en modo cliente.

Antes de empezar

Asegúrese de que el gestor de colas ya existe y se ha iniciado. Determine si ya se han habilitado los registros de autenticación de canal emitiendo el mandato MQSC:

```
DISPLAY QMGR CHLAUTH
```

Importante: Esta tarea espera a que se habiliten los registros de autenticación de canal. Si se trata de un gestor de colas utilizado por otros usuarios y aplicaciones, cambiar este valor afectará a los demás usuarios y aplicaciones. Si su gestor de colas no utiliza los registros de autenticación de canal, se puede sustituir el paso 4 por un método de autenticación alternativo, tal como una salida de seguridad, que establece MCAUSER en el *non-privileged-user-id* que obtendrá en el paso “1” en la página 1195.

Debe saber el nombre de canal que la aplicación espera utilizar para que se le pueda permitir el uso del canal. También debe saber qué objetos, por ejemplo, colas o temas, espera utilizar la aplicación para que se le pueda permitir utilizarlos.

Acerca de esta tarea

Esta tarea crea un ID de usuario sin privilegios para utilizarlo con una aplicación cliente que se conecta al gestor de colas. El acceso se otorga solo a la aplicación de cliente para que pueda utilizar el canal que necesita y la cola que necesita mediante este ID de usuario.

Procedimiento

1. Obtenga un ID de usuario en el sistema en el que se ejecuta el gestor de colas.

   En esta tarea, este ID de usuario no debe ser el de un usuario administrativo con privilegios. Este ID de usuario es la autorización bajo la cual se ejecutará la conexión de cliente en el gestor de colas.
2. Inicie una sesión de escucha.
   a) Asegúrese de que se ha iniciado el iniciador de canal. Si no es así, inícielo emitiendo el mandato **START CHINIT** .
   b) Inicie el programa de escucha con el mandato siguiente:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   donde *nnnn* es el número de puerto que ha elegido.
3. Si la aplicación utiliza SYSTEM.DEF.SVRCONN>, este canal ya se ha definido. Si la aplicación utiliza otro canal, créelo con el mandato MQSC:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

channel-name es el nombre del canal.

4. Cree una norma de autenticación de canal que permita que solo la dirección IP del sistema cliente utilice el canal con el mandato MQSC:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

donde

channel-name es el nombre del canal.
client-machine-IP-address es la dirección IP del sistema cliente. Si su aplicación de cliente de ejemplo se ejecuta en la misma máquina que el gestor de colas, utilice una dirección IP de '127.0.0.1', si su aplicación se va a conectar utilizando 'localhost'. Si se van a conectar varias máquinas diferentes, puede utilizar un patrón o un rango en lugar de una única dirección IP. Para obtener más información, consulte la sección Direcciones IP genéricas.
non-privileged-user-id es el ID de usuario que ha obtenido en el paso "1" en la página 1195

5. Si la aplicación utiliza SYSTEM.DEFAULT.LOCAL.QUEUE, esta cola ya está definida. Si la aplicación utiliza otra cola, créela con el mandato MQSC:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

donde *queue-name* es el nombre de su cola.

6. Otorgue acceso para conectar con el gestor de colas y consultarlo:
   a) Asegúrese de que se ha iniciado el iniciador de canal. Si no lo está, inícielo con el mandato START CHINIT.
   b) Inicie un escucha de TCP, por ejemplo, escriba este mandato:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   donde *nnnn* es el número de puerto que ha elegido.

7. Si la aplicación es una aplicación de punto a punto, es decir, utiliza las colas, conceda acceso para que se puedan realizar consultas, transferir y obtener mensajes utilizando su cola con el ID de usuario que se ha de utilizar, mediante los mandatos MQSC:

   Emita los mandatos RACF:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

donde

qmgr-name es el nombre del gestor de colas
queue-name es el nombre de la cola.
non-privileged-user-id es el ID de usuario que ha obtenido en el paso "1" en la página 1195

8. Si la aplicación es una aplicación de publicación/suscripción, es decir hace uso de temas, conceda acceso para permitir la publicación y suscripción utilizando su tema con el ID de usuario que se ha de utilizar, mediante los mandatos RACF:

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

donde

qmgr-name es el nombre del gestor de colas
non-privileged-user-id es el ID de usuario que ha obtenido en el paso "1" en la página 1195
De este modo, el *non-privileged-user-id* tendrá acceso a cualquier tema del árbol de temas, o puede definir un objeto de tema utilizando **DEFINE TOPIC** y otorgar accesos únicamente a la parte del árbol de temas a la que hace referencia dicho objeto de tema. Para obtener más información, consulte Control del acceso de usuario a temas.

Qué hacer a continuación

La aplicación cliente se puede ahora conectar al gestor de colas y transferir u obtener mensajes utilizando la cola.

Conceptos relacionados

z/OS Autorización para trabajar con objetos IBM MQ en z/OS

Referencia relacionada

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

z/OS *Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS*

Para preparar una aplicación de ejemplo que ejecute en el entorno de proceso por lotes, siga los mismos pasos que al crear cualquier aplicación por lotes de IBM MQ for z/OS.

Estos pasos se listan en "Creación de aplicaciones por lotes de z/OS" en la página 1050.

De forma alternativa, donde se proporcione un formato ejecutable de un ejemplo, se puede ejecutar desde la biblioteca de carga thlqual.SCSQLOAD.

Nota: La versión en lenguaje ensamblador del ejemplo Examinar utiliza bloques de control de datos (DCB), por lo que debe editarlo mediante enlaces utilizando `RMODE(24)`.

Los miembros de biblioteca que hay que usar se listan en Tabla 172 en la página 1198, Tabla 173 en la página 1198, Tabla 174 en la página 1199 y Tabla 175 en la página 1199.

Debe editar el JCL de ejecución para los ejemplos que desee utilizar (consulte Tabla 172 en la página 1198, Tabla 173 en la página 1198, Tabla 174 en la página 1199 y Tabla 175 en la página 1199).

La sentencia PARM en el JCL proporcionado contiene una serie de parámetros que hay que modificar. Para ejecutar los programas de ejemplo en C, separe los parámetros con espacios; para ejecutar los programas de ejemplo en ensamblador, COBOL y PL/I, sepárelos con comas. Por ejemplo, si el nombre del gestor de colas es CSQ1 y desea ejecutar la aplicación con una cola llamada LOCALQ1, en el JCL de COBOL, PL/I y ensamblador, la sentencia PARM debe tener el aspecto siguiente:

```
PARM=(CSQ1,LOCALQ1)
```

En el JCL de C, la sentencia PARM debe tener un aspecto similar al siguiente:

```
PARM=('CSQ1 LOCALQ1')
```

Ahora ya está preparado para someter los trabajos.

z/OS *Nombres de las aplicaciones de proceso por lotes de ejemplo sobre z/OS*

Resumen de los programas que se suministran para las aplicaciones de proceso por lotes de ejemplo.

Las tablas siguientes resumen los programas de aplicación de proceso por lotes:

- Tabla 172 en la página 1198 Programas de ejemplos de transferencia y obtención
- Tabla 173 en la página 1198 Programa de ejemplo de examen

- Tabla 174 en la página 1199 Programa de ejemplo para imprimir mensaje
- Tabla 175 en la página 1199 Programas de ejemplo de publicación/suscripción
- Tabla 176 en la página 1199Otros ejemplos

Tabla 172. Programas de ejemplos transferencia y obtención de proceso por lotes

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4BCJ1	C	Programa fuente de obtención	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Programa fuente de transferencia	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	JCL de ejecución de ejemplo para CSQ4BCJ1 y CSQBCK1	SCSQPROC	Ninguna
CSQ4BVJ1	COBOL	Programa fuente de obtención	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Programa fuente de transferencia	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	JCL de ejecución de ejemplo para CSQBVJ1 y CSQBVK1	SCSQPROC	Ninguna

Tabla 173. Programa de ejemplo de proceso por lotes de examen

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4BVA1	COBOL	Programa fuente	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	JCL de ejecución de ejemplo para CSQ4BVA1	SCSQPROC	Ninguna
CSQ4BAA1	Ensamblador	Programa fuente	SCSQASMS	SCSQLOAD
CSQ4BAAR	Ensamblador	JCL de ejecución de ejemplo para CSQ4BAA1	SCSQPROC	Ninguna
CSQ4BCA1	C	Programa fuente	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	JCL de ejecución de ejemplo para CSQ4BCA1	SCSQPROC	Ninguna
CSQ4BPA1	PL/I	Programa fuente	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	JCL de ejecución de ejemplo para CSQ4BPA1	SCSQPROC	Ninguna

Tabla 174. Programa de ejemplo de proceso por lotes para imprimir mensaje (sólo lenguaje C)

Nombre del miembro	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4BCG1	Programa fuente	SCSQC37S	SCSQLOAD
CSQ4BCGR	JCL de ejecución de ejemplo para CSQ4BCG1	SCSQPROC	Ninguna
CSQ4BCL1	Programa fuente para examinar	SCSQC37S	SCSQLOAD
CSQ4BCLR	JCL de ejecución de ejemplo para CSQ4BCL1	SCSQPROC	Ninguna

Tabla 175. Programas de ejemplos para publicar/suscribir

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	JCL en SCSQPROC	Archivo ejecutable proporcionado en la biblioteca
CSQ4BCP1	C	Programa fuente para publicar en un tema	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	Programa fuente para subscribirse a un tema y obtener mensajes	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	Programa fuente para suscribirse a un tema utilizando un destino proporcionado por el usuario y obtener mensajes	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Programa fuente para suscribirse a un tema utilizando opciones ampliadas y obtener mensajes	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	Programa fuente para publicar en un tema	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	Programa fuente para subscribirse a un tema y obtener mensajes	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Tabla 176. Otros ejemplos

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	JCL en SCSQPROC	Archivo ejecutable proporcionado en la biblioteca
CSQ4BCS1	C	Programa fuente de consumo asíncrono	SCSQC37S	CSQ4BCSC	SCSQLOAD

Tabla 176. Otros ejemplos (continuación)

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	JCL en SCSQPROC	Archivo ejecutable proporcionado en la biblioteca
CSQ4BCS2	C	Programa fuente de transferencia asíncrona y comprobar estado	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Programa fuente para consultar propiedades del mensaje	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	Programa fuente para establecer propiedades del mensaje	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *Preparación de aplicaciones de ejemplo para el entorno TSO en z/OS*

Para preparar una aplicación de ejemplo que ejecute en el entorno TSO, siga los mismos pasos que al crear cualquier aplicación por lotes de IBM MQ for z/OS.

Estos pasos se listan en "Creación de aplicaciones por lotes de z/OS" en la página 1050. Los miembros de biblioteca que hay que usar se listan en Tabla 177 en la página 1200.

De forma alternativa, donde se proporcione un formato ejecutable de un ejemplo, se puede ejecutar desde la biblioteca de carga thlqual.SCSQLOAD.

En la aplicación de ejemplo de gestor de correo, asegúrese de que las colas que utiliza están disponibles en el sistema. Están definidas en el miembro **thlqual**.SCSQPROC(CSQ4CVD). Para asegurarse de que estas colas estén siempre disponibles, se pueden añadir estos miembros al conjunto de datos de entrada de inicialización CSQINP2 o utilizar el programa CSQUTIL para cargar estas definiciones de cola.

z/OS *Nombres de las aplicaciones TSO de ejemplo en z/OS*

Información sobre los nombres de los programas que se suministran para cada una de las aplicaciones TSO de ejemplo, y las bibliotecas donde residen el origen, el JCL y, solo para el ejemplo de Manejador de mensajes, los archivos ejecutables.

Los programas de aplicación TSO se resumen en las tablas siguientes:

- Ejemplo de gestor de correo de Tabla 177 en la página 1200
- Ejemplo de manejador de mensajes de Tabla 178 en la página 1201

Estos ejemplos utilizan paneles ISPF. Por lo tanto, debe incluir el apéndice de ISPF, ISPLINK, cuando edita mediante enlaces los programas.

Tabla 177. Ejemplo de gestor de correo de TSO

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4CVD	independiente	Definiciones de objeto de IBM MQ for z/OS	SCSQPROC
CSQ40	independiente	Mensajes de ISPF	SCSQMSGE

Tabla 177. Ejemplo de gestor de correo de TSO (continuación)

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4RVD1	COBOL	CLIST para iniciar CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Programa fuente para el programa Menu	SCSQCOBS
CSQ4TVD2	COBOL	Programa fuente para el programa Get Mail	SCSQCOBS
CSQ4TVD4	COBOL	Programa fuente para el programa Send Mail	SCSQCOBS
CSQ4TVD5	COBOL	Programa fuente para el programa Nickname	SCSQCOBS
CSQ4VDP1-6	COBOL	Definiciones de paneles	SCSQPNLA
CSQ4VD0	COBOL	Definición de datos	SCSQCOBC
CSQ4VD1	COBOL	Definición de datos	SCSQCOBC
CSQ4VD2	COBOL	Definición de datos	SCSQCOBC
CSQ4VD4	COBOL	Definición de datos	SCSQCOBC
CSQ4RCD1	C	CLIST para iniciar CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Programa fuente para el programa Menu	SCSQC37S
CSQ4TCD2	C	Programa fuente para el programa Get Mail	SCSQC37S
CSQ4TCD4	C	Programa fuente para el programa Send Mail	SCSQC37S
CSQ4TCD5	C	Programa fuente para el programa Nickname	SCSQC37S
CSQ4CDP1-6	C	Definiciones de paneles	SCSQPNLA
CSQ4TC0	C	Archivo de inclusión	SCSQC370

Tabla 178. Ejemplo de manejador de mensajes de TSO

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4TCH0	C	Definición de datos	SCSQC370	Ninguna
CSQ4TCH1	C	Programa fuente	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Programa fuente	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	Programa fuente	SCSQC37S	SCSQLOAD
CSQ4RCH1	C y COBOL	CLIST para iniciar CSQ4TCH1 o CSQ4TVH1	SCSQCLST	Ninguna

Tabla 178. Ejemplo de manejador de mensajes de TSO (continuación)				
Nombre del miembro	**Para el lenguaje**	**Descripción**	**Archivo de origen proporcionado en la biblioteca**	**Archivo ejecutable proporcionado en la biblioteca**
CSQ4CHP1	C y COBOL	Definición de panel	SCSQPNLA	Ninguna
CSQ4CHP2	C y COBOL	Definición de panel	SCSQPNLA	Ninguna
CSQ4CHP3	C y COBOL	Definición de panel	SCSQPNLA	Ninguna
CSQ4CHP9	C y COBOL	Definición de panel	SCSQPNLA	Ninguna
CSQ4TVH0	COBOL	Definición de datos	SCSQCOBC	Ninguna
CSQ4TVH1	COBOL	Programa fuente	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Programa fuente	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Programa fuente	SCSQCOBS	SCSQLOAD

z/OS *Preparación de las aplicaciones de ejemplo para el entorno CICS en z/OS*

Antes de ejecutar los programas de ejemplo de CICS, inicie sesión en CICS utilizando un LOGMODE de 32702. Esto se debe a que los programas de ejemplo se han escrito para utilizar una pantalla 3270 en modo 2.

Para preparar una aplicación de ejemplo que que se ejecute en el entorno de CICS, realice los pasos siguientes:

1. Cree la correlación de descripciones simbólicas y la correlación de pantalla física del ejemplo ensamblando los fuentes de definición de pantalla BMS (proporcionados en la biblioteca **thlqual**.SCSQMAPS, donde **thlqual** es el calificador de alto nivel utilizado por la instalación). Cuando haya nombrado las correlaciones, use el nombre de los fuentes de definición de pantalla BMS (no disponibles para los programas de ejemplo de colocación y obtención), pero omita el último carácter de dicho nombre.
2. Realice los mismos pasos que haría al crear cualquier aplicación CICS IBM MQ for z/OS. Estos pasos se listan en “Creación de aplicaciones CICS en z/OS” en la página 1053. Los miembros de biblioteca que hay que usar se listan en Tabla 179 en la página 1203, Tabla 180 en la página 1203, Tabla 181 en la página 1204 y Tabla 182 en la página 1204.

   De forma alternativa, si se proporciona una formato ejecutable de un ejemplo, se puede ejecutar desde la librería de carga thlqual.SCSQCICS.
3. Identifique el conjunto de correlaciones, los programas y la transacción en CICS actualizando el conjunto de datos de definición del sistema de CICS (CSD). Las definiciones necesarias se encuentran en el miembro **thlqual**.SCSQPROC (CSQ4S100). Para obtener ayuda sobre cómo hacerlo, consulte la sección *El adaptador CICS-IBM MQ* en la documentación del producto CICS Transaction Server for z/OS 4.1 en: CICS Transaction Server for z/OS 4.1, El adaptador CICS-IBM MQ.

   Nota: En la aplicación de ejemplo de comprobación de crédito, se obtiene un mensaje de error en esta etapa si todavía no se ha creado el conjunto de datos VSAM que se utiliza en dicho ejemplo.
4. En las aplicaciones de ejemplo de comprobación de crédito y gestor de correo, asegúrese de que las colas que utilizan estén disponibles en el sistema. Para el ejemplo de comprobación de crédito, se definen en el miembro **thlqual**.SCSQPROC (CSQ4CVB) para COBOL y **thlqual**.SCSQPROC (CSQ4CCB) para C. Para el ejemplo de gestor de correo, se definen en el miembro **thlqual**.SCSQPROC (CSQ4CVD). Para asegurarse de que estas colas estén siempre disponibles, se pueden añadir estos miembros al conjunto de datos de entrada de inicialización CSQINP2 o utilizar el programa CSQUTIL para cargar estas definiciones de cola.

   En la aplicación de ejemplo de atributos de cola, se pueden utilizar una o varias de las colas que se suministran para las otras aplicaciones de ejemplo. De forma alternativa, puede utilizar sus propias

colas. Sin embargo, en el formato en que se proporciona, este ejemplo solo funciona con las colas que tienen los caracteres CSQ4SAMP en los ocho primeros bytes de su nombre.

z/OS *Nombres de las aplicaciones CICS de ejemplo en z/OS*

En este tema se proporciona un resumen de los programas proporcionados para las aplicaciones de ejemplo de CICS.

Los programas de aplicación de CICS se resumen en las tablas siguientes:

- Tabla 179 en la página 1203 Programas de ejemplos de transferencia y obtención
- Tabla 180 en la página 1203 - Ejemplo de atributos de cola
- Tabla 181 en la página 1204 - Ejemplo Mail Manager (Gestor de correo) (solo COBOL)
- Tabla 182 en la página 1204 - Ejemplo de comprobación de crédito
- Tabla 183 en la página 1205 - Ejemplos de consumo asíncrono y publicación/suscripción

Tabla 179. CICS Programas de ejemplos de transferencia y obtención

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4CCK1	C	Programa fuente de transferencia	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Programa fuente de obtención	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Programa fuente de obtención	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Programa fuente de transferencia	SCSQCOBS	SCSQCICS
CSQ4S100	independiente	Conjunto de datos de definición del sistema CICS	SCSQPROC	Ninguna

Tabla 180. CICS - Ejemplo de atributos de cola

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4CVC1	COBOL	Programa fuente	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Definición de mensajes	SCSQCOBC	Ninguna
CSQ4VCMS	COBOL	Definición de pantalla BMS	SCSQMAPS	SCSQCICS (CSQ4ACM con nombre)
CSQ4CAC1	Ensamblador	Programa fuente	SCSQASMS	SCSQCICS
CSQ4AMSG	Ensamblador	Definición de mensajes	SCSQMACS	Ninguna
CSQ4ACMS	Ensamblador	Definición de pantalla BMS	SCSQMAPS	SCSQCICS (CSQ4ACM con nombre)
CSQ4CCC1	C	Programa fuente	SCSQC37S	SCSQCICS

Tabla 180. CICS - Ejemplo de atributos de cola (continuación)

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca	Archivo ejecutable proporcionado en la biblioteca
CSQ4CMSG	C	Definición de mensajes	SCSQC370	Ninguna
CSQ4CCMS	C	Definición de pantalla BMS	SCSQMAPS	SCSQCICS (CSQ4ACM con nombre)
CSQ4S100	independiente	Conjunto de datos de definición del sistema CICS	SCSQPROC	Ninguna

Tabla 181. CICS - Ejemplo Mail Manager (Gestor de correo) (solo COBOL)

Nombre del miembro	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4CVD	Definiciones de objeto de IBM MQ for z/OS	SCSQPROC
CSQ4CVD1	Fuente para el programa Menu	SCSQCOBS
CSQ4CVD2	Fuente para el programa Get Mail	SCSQCOBS
CSQ4CVD3	Fuente para el programa Display Message	SCSQCOBS
CSQ4CVD4	Código fuente para el programa Send Mail	SCSQCOBS
CSQ4CVD5	Fuente para el programa Nickname	SCSQCOBS
CSQ4VDMS	Fuente de definición de pantalla BMS	SCSQMAPS
CSQ4S100	Conjunto de datos de definición del sistema CICS	SCSQPROC
CSQ4VD0	Definición de datos	SCSQCOBC
CSQ4VD3	Definición de datos	SCSQCOBC
CSQ4VD4	Definición de datos	SCSQCOBC

Tabla 182. CICS - Ejemplo de comprobación de crédito

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4CVB	independiente	Definiciones de objeto de IBM MQ	SCSQPROC
CSQ4CCB	independiente	Definiciones de objeto de IBM MQ	SCSQPROC
CSQ4CVB1	COBOL	Fuente para el programa de interfaz de usuario	SCSQCOBS
CSQ4CVB2	COBOL	Fuente para el gestor de solicitud de crédito	SCSQCOBS

Tabla 182. CICS - Ejemplo de comprobación de crédito (continuación)

Nombre del miembro	Para el lenguaje	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4CVB3	COBOL	Fuente para el programa de cuenta	SCSQCOBS
CSQ4CVB4	COBOL	Fuente para el programa de distribución	SCSQCOBS
CSQ4CVB5	COBOL	Fuente para el programa de consulta de agencia	SCSQCOBS
CSQ4CCB1	C	Fuente para el programa de interfaz de usuario	SCSQC37S
CSQ4CCB2	C	Fuente para el gestor de solicitud de crédito	SCSQC37S
CSQ4CCB3	C	Fuente para el programa de cuenta	SCSQC37S
CSQ4CCB4	C	Fuente para el programa de distribución	SCSQC37S
CSQ4CCB5	C	Fuente para el programa de consulta de agencia	SCSQC37S
CSQ4CB0	C	Archivo de inclusión	SCSQC370
CSQ4CBMS	C	Fuente de definición de pantalla BMS	SCSQMAPS
CSQ4VBMS	COBOL	Fuente de definición de pantalla BMS	SCSQMAPS
CSQ4VB0	COBOL	Definición de datos	SCSQCOBC
CSQ4VB1	COBOL	Definición de datos	SCSQCOBC
CSQ4VB2	COBOL	Definición de datos	SCSQCOBC
CSQ4VB3	COBOL	Definición de datos	SCSQCOBC
CSQ4VB4	COBOL	Definición de datos	SCSQCOBC
CSQ4VB5	COBOL	Definición de datos	SCSQCOBC
CSQ4VB6	COBOL	Definición de datos	SCSQCOBC
CSQ4VB7	COBOL	Definición de datos	SCSQCOBC
CSQ4VB8	COBOL	Definición de datos	SCSQCOBC
CSQ4BAQ	independiente	Fuente para el conjunto de datos VSAM	SCSQPROC
CSQ4FILE	independiente	JCL para crear un conjunto de datos VSAM utilizado por CSQ4CVB3	SCSQPROC
CSQ4S100	independiente	Conjunto de datos de definición del sistema CICS	SCSQPROC

Tabla 183. CICS - Ejemplos de consumo asíncrono y publicación/suscripción

Nombre del miembro	Descripción	Archivo de origen proporcionado en la biblioteca
CSQ4CVCN	Fuente para el programa Simple Message Consumption (Consumo de mensajes sencillos)	SCSQCOBS

Tabla 183. CICS - Ejemplos de consumo asíncrono y publicación/suscripción (continuación)		
Nombre del miembro	**Descripción**	**Archivo de origen proporcionado en la biblioteca**
CSQ4CVCT	Fuente para el programa Control Message Consumption (Consumo de mensajes de control)	SCSQCOBS
CSQ4CVEV	Código fuente para el programa Event Handler (Manejador de suceso)	SCSQCOBS
CSQ4CVPT	Fuente para el programa Message Put Client (Cliente Put de mensajes)	SCSQCOBS
CSQ4CVRG	Fuente para el programa Registration Client (Cliente de registro)	SCSQCOBS
CSQ4S100	Conjunto de datos de definición del sistema CICS	SCSQPROC

z/OS *Preparación de la aplicación de ejemplo para el entorno IMS en z/OS*

Una parte de la aplicación de ejemplo de comprobación de crédito puede ejecutar en un entorno IMS.

Para preparar esta parte de la aplicación a fin de que ejecute con el ejemplo de CICS, siga antes los pasos descritos en "Preparación de las aplicaciones de ejemplo para el entorno CICS en z/OS" en la página 1202.

Luego siga estos pasos:

1. Realice los mismos pasos que haría al crear cualquier aplicación IMS IBM MQ for z/OS. Estos pasos se listan en "Creación de aplicaciones IMS (BMP o MPP)" en la página 1054. Los miembros de biblioteca que hay que usar se listan en Tabla 184 en la página 1206.
2. Identifique el programa de aplicación y la base de datos en IMS. Los ejemplos se proporcionan con PSBGEN, DBDGEN, la definición ACB y las sentencias IMSGEN y IMSDALOC para habilitar esto.
3. Cargue la base de datos CSQ4CA personalizando y ejecutando el JCL de ejemplo proporcionado a tal fin (CSQ4ILDB). Este JCL carga la base de datos con datos del archivo CSQ4BAQ. Actualice la región de control IMS con una sentencia DD para la base de datos CSQ4CA.
4. Inicie el programa de cuenta de ahorros como un programa de procesamiento de mensajes por lotes (BMP) personalizando y ejecutando el JCL de ejemplo proporcionado a tal fin. Este JCL inicia un programa BMP orientado a lotes. Para ejecutar el programa como un programa BMP orientado a mensajes, elimine los caracteres de comentario de línea del JCL que contiene la sentencia IN=.

z/OS *Nombres de la aplicación IMS de ejemplo en z/OS*

Esta información proporciona una tabla con la lista de los orígenes y JCL que se suministran para la aplicación IMS de ejemplo de comprobación de crédito (Credit Check).

Tabla 184. Fuente y JCL para el ejemplo Comprobación de crédito IMS (sólo C)		
Nombre del miembro	**Descripción**	**Suministrado en la biblioteca**
CSQ4CVB	Definiciones de objeto de IBM MQ	SCSQPROC
CSQ4ICB3	Fuente para el programa de cuenta	SCSQC37S

Tabla 184. Fuente y JCL para el ejemplo Comprobación de crédito IMS (sólo C) (continuación)		
Nombre del miembro	**Descripción**	**Suministrado en la biblioteca**
CSQ4ICBL	Fuente para la carga de la base de datos de la cuenta	SCSQC37S
CSQ4CBI	Definición de datos	SCSQC370
CSQ4PSBL	PSBGEN JCL para el programa de carga de base de datos	SCSQPROC
CSQ4PSB3	PSBGEN JCL para el programa de cuenta	SCSQPROC
CSQ4DBDS	DBDGEN JCL para base de datos CSQ4CA	SCSQPROC
CSQ4GIMS	Definiciones de macro GEN de IMS para CSQ4IVB3 y CSQ4CA	SCSQPROC
CSQ4ACBG	Definición de bloque de control de aplicación (ACB) para CSQ4IVB3	SCSQPROC
CSQ4BAQ	Fuente para base de datos	SCSQPROC
CSQ4ILDB	JCL de ejecución de ejemplo para trabajo de carga de base de datos	SCSQPROC
CSQ4ICBR	JCL de ejecución de ejemplo para programa de cuenta	SCSQPROC
CSQ4DYNA	Definiciones macro DALOC de IMS para base de datos	SCSQPROC

z/OS *Ejemplos de Put en z/OS*

Los programas de ejemplo Put ponen mensajes en una cola utilizando la llamada MQPUT.

Los programas fuente se suministran en C y COBOL en los entornos de proceso por lotes y CICS (consulte también Tabla 172 en la página 1198 y Tabla 179 en la página 1203).

Diseño del programa de ejemplo Put

El flujo de la lógica del programa es:

1. Se conecta con el gestor de colas utilizando la llamada MQCONN. Si esta llamada falla, se imprimen los códigos de terminación y de razón, y se detiene el proceso.

   Nota: Si está ejecutando el ejemplo en un entorno de CICS, no es necesario emitir una llamada MQCONN; si lo hace, devuelve DEF_HCONN. Puede utilizar el descriptor de conexión MQHC_DEF_HCONN en las llamadas MQI siguientes.
2. Abra la cola utilizando la llamada MQOPEN con la opción MQOO_OUTPUT. En la entrada de esta llamada, el programa usa el descriptor de conexión devuelto en el paso “1” en la página 1209. En la estructura del descriptor de objeto (MQOD), utiliza los valores predeterminados en todos los campos excepto el campo de nombre de cola, que se pasa como un parámetro al programa. Si la llamada MQOPEN falla, se imprimen los códigos de terminación y de razón, y se detiene el procesamiento.
3. Crear un bucle en el programa, emitiendo llamadas MQPUT hasta que se pongan en la cola el número necesario de mensajes. Si una llamada MQPUT falla, el bucle se abandona anticipadamente, y no se intentan más llamadas MQPUT; se devuelven los códigos de terminación y de razón.
4. Se cierra la cola con la llamada MQCLOSE empleando el descriptor de objeto devuelto en el paso “2” en la página 1209. Si esta llamada falla, se imprimen los códigos de terminación y de razón.

5. Se desconecta del gestor de colas con la llamada MQDISC empleando el descriptor de conexión devuelto en el paso “1” en la página 1209. Si esta llamada falla, se imprimen los códigos de terminación y de razón.

   Nota: Si está ejecutando el ejemplo en un entorno de CICS, no es necesario emitir una llamada MQDISC.

z/OS *Ejemplos de colocación para el entorno de proceso por lotes de z/OS*

Utilice este tema cuando considere ejemplos de colocación para el entorno de proceso por lotes.

Para ejecutar los ejemplos, edite y ejecute el JCL de ejemplo, tal y como se describe en “Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS” en la página 1197.

Los programas reicben los parámetros siguientes en un EXEC PARM, separados por espacios en C y por comas en COBOL:

1. El nombre del gestor de colas (4 caracteres).
2. El nombre de la cola de destino (48 caracteres).
3. El número de mensajes (hasta 4 dígitos).
4. El carácter de relleno que se escribe en el mensaje (1 carácter).
5. El número de caracteres que se escriben el mensaje (hasta 4 dígitos).
6. La persistencia del mensaje (1 carácter: P para persistente o N para no persistente).

Si escribe incorrectamente alguno de estos parámetros, recibirá los correspondientes mensajes de error.

Todos los mensajes de los ejemplos se escriben en el conjunto de datos SYSPRINT.

Notas de uso

- Para mantener la simplicidad de los ejemplos, hay algunas diferencias funcionales menores entre las distintas versiones de lenguaje. Sin embargo, estas diferencias se minimizan si se utiliza el diseño de parámetros que se muestra en el JCL de ejecución de ejemplo, CSQ4BCJR y CSQ4BVJR. Ninguna de las diferencias tiene que ver con la MQI.
- CSQ4BCK1 permite especificar más de cuatro dígitos para el número de mensajes enviados y la longitud de los mismos.
- En los dos campos numéricos, especifique cualquier dígito comprendido en el rango de 1 a 9999. El valor que especifique tiene que ser un número positivo. Por ejemplo, para colocar un solo mensaje, se pueden especificar los valores 1, 01, 001 o 0001. Si se especifican valores no numéricos o negativos, podría dar error. Por ejemplo, si especifica -1, el programa en COBOL envía un mensaje de 1 byte, pero el programa en C recibe un error.
- En ambos programas, CSQ4BCK1 y CSQ4BVK1, hay que especificar P en el parámetro de persistencia, ++PER++, si se desea que el mensaje sea persistente. Si no se hace, el mensaje no será persistente.

z/OS *Ejemplos de colocación para el entorno CICS en z/OS*

Utilice este tema cuando considere ejemplos de colocación para el entorno CICS.

Las transacciones reciben los parámetros siguientes, separados por comas:

1. El número de mensajes (hasta 4 dígitos).
2. El carácter de relleno que se escribe en el mensaje (1 carácter).
3. El número de caracteres que se escriben el mensaje (hasta 4 dígitos).
4. La persistencia del mensaje (1 carácter: P para persistente o N para no persistente).
5. El nombre de la cola de destino (48 caracteres).

Si escribe incorrectamente alguno de estos parámetros, recibirá los correspondientes mensajes de error.

Para el ejemplo en COBOL, invoque el ejemplo de colocación en el entorno CICS especificando:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

Para el ejemplo en C, inboque el ejemplo de colocación en el entorno CICS especificando:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Los mensajes de los ejemplos se visualizan en la pantalla.

Notas de uso

- Para mantener la simplicidad de los ejemplos, hay algunas diferencias funcionales menores entre las distintas versiones de lenguaje. Ninguna de las diferencias tiene que ver con la MQI.
- Si se especifica un nombre de cola que tenga más de 48 caracteres, su longitud se truncará al máximo de 48 caracteres, pero no se devolverá ningún mensaje de error.
- Antes de entrar en la transacción, pulse la tecla CLEAR.
- En los dos campos numéricos, especifique cualquier dígito comprendido en el rango de 1 a 9999. El valor que especifique tiene que ser un número positivo. Por ejemplo, para colocar un único mensaje, se pueden especificar los valores 1, 01, 001 o 0001. Si se especifican valores no numéricos o negativos, podría dar error. Por ejemplo, si se especifica -1, el programa COBOL envía un mensaje de 1 byte y el programa C termina de forma anómala con un error de malloc().
- En ambos programas, CSQ4CCK1 y CSQ4CVK1, especifique P en el parámetro de persistencia si desea que el mensaje sea persistente. En el caso de los mensajes no persistentes, especifique N en el parámetro de persistencia. Si especifica cualquier otro valor, recibirá un mensaje de error.
- Los mensajes se colocan en punto de sincronización porque se usan los valores predeterminados en todos los parámetros excepto aquellos establecidos durante la invocación del programa.

z/OS *Ejemplos de obtención en z/OS*

Los programas de ejemplo de obtención obtienen los mensajes de una cola utilizando la llamada MQGET.

Los programas fuente se suministran en C y COBOL en los entornos de proceso por lotes y CICS (consulte también Tabla 172 en la página 1198 y Tabla 179 en la página 1203).

z/OS *Diseño del ejemplo de obtención en z/OS*

Obtenga más información sobre el diseño del ejemplo de obtención y de algunas notas de uso que se deben tener en cuenta.

El flujo de la lógica del programa es:

1. Se conecta con el gestor de colas utilizando la llamada MQCONN. Si esta llamada falla, se imprimen los códigos de terminación y de razón, y se detiene el proceso.

   Nota: Si está ejecutando el ejemplo en un entorno de CICS, no es necesario emitir una llamada MQCONN; si lo hace, devuelve DEF_HCONN. Puede utilizar el descriptor de conexión MQHC_DEF_HCONN en las llamadas MQI siguientes.
2. Abrir la cola usando la llamada MQOPEN con las opciones MQOO_INPUT_SHARED y MQOO_BROWSE. En la entrada de esta llamada, el programa usa el descriptor de conexión devuelto en el paso “1” en la página 1209. En la estructura del descriptor de objeto (MQOD), utiliza los valores predeterminados en todos los campos excepto el campo de nombre de cola, que se pasa como un parámetro al programa. Si la llamada MQOPEN falla, se imprimen los códigos de terminación y de razón, y se detiene el procesamiento.
3. Crear un bucle dentro del programa que emita llamadas MQGET hasta que se recupere el número necesario de mensajes de la cola. Si una llamada MQGET falla, se sale del bucle antes de tiempo, no se intentan más llamadas MQGET y se devuelven los códigos de terminación y de razón. En la llamada MQGET se especifican las opciones siguientes:

- MQGMO_NO_WAIT
- MQGMO_ACCEPT_TRUNCATED_MESSAGE
- MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT
- MQGMO_BROWSE_FIRST y MQGMO_BROWSE_NEXT

Para obtener una descripción de estas opciones, consulte MQGET. Por cada mensaje, se imprime su número de mensaje, seguido de su longitud y sus datos.

4. Se cierra la cola con la llamada MQCLOSE empleando el descriptor de objeto devuelto en el paso "2" en la página 1209. Si esta llamada falla, se imprimen los códigos de terminación y de razón.
5. Se desconecta del gestor de colas con la llamada MQDISC empleando el descriptor de conexión devuelto en el paso "1" en la página 1209. Si esta llamada falla, se imprimen los códigos de terminación y de razón.

   Nota: Si está ejecutando el ejemplo en un entorno de CICS, no es necesario emitir una llamada MQDISC.

Notas de uso

- Para mantener la simplicidad de los ejemplos, hay algunas diferencias funcionales menores entre las distintas versiones de lenguaje. Sin embargo, estas diferencias se minimizan si se utiliza el diseño de parámetros que se muestra en el JCL de ejecución de ejemplo, CSQ4BCJR y CSQ4BVJR. Ninguna de las diferencias tiene que ver con la MQI.
- CSQ4BCJ1 permite especificar más de cuatro dígitos en el número de mensajes recuperados.
- Los mensajes de más de 64 KB se truncan.
- CSQ4BCJ1 solo puede mostrar correctamente mensajes de caracteres, porque solo visualiza hasta el primer carácter NULL (\0).
- En el campo numérico numeric number-of-messages, especifique cualquier dígito en el rango de 1 a 9999. El valor que especifique tiene que ser un número positivo. Por ejemplo, para obtener un solo mensaje, se pueden especificar los valores 1, 01, 001 o 0001. Si se especifican valores no numéricos o negativos, podría dar error. Por ejemplo, si se especifica -1, el programa en COBOL recupera un mensaje, pero el programa en C no recupera ningún mensaje.
- En ambos programas, CSQ4BCJ1 y CSQ4BVJ1, especifique B en el parámetro de obtención, ++GET++, si desea examinar los mensajes.
- En ambos programas, CSQ4BCJ1 y CSQ4BVJ1, especifique S en el parámetro punto de sincronización, ++SYNC ++, para que los mensajes se recuperen en el punto de sincronización.

Ejemplos de obtención para el entorno de proceso por lotes de z/OS

Para ejecutar los ejemplos, edite y ejecute el JCL de ejemplo, tal y como se describe en "Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS" en la página 1197.

Los programas reicben los parámetros siguientes en un EXEC PARM, separados por espacios en C y por comas en COBOL:

1. El nombre del gestor de colas (4 caracteres).
2. El nombre de la cola de destino (48 caracteres).
3. El número de mensajes que se van a obtener (hasta cuatro dígitos).
4. La opción de examinar/obtener mensaje (un carácter: B para examinar o D para obtener de forma destructiva los mensajes).
5. El control de punto de sincronización (un carácter: S para que haya punto de sincronización o N para que no haya ningún punto de sincronización).

Si escribe incorrectamente alguno de estos parámetros, recibirá los correspondientes mensajes de error.

La salida de los ejemplos se escribe en el conjunto de datos SYSPRINT:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *Ejemplos de obtención para el entorno CICS en z/OS*

Consideraciones especiales para los ejemplos de obtención para el entorno CICS.

Las transacciones reciben los parámetros siguientes en un EXEC PARM, separados por comas:

1. El número de mensajes que se van a obtener (hasta cuatro dígitos).
2. La opción de examinar/obtener mensaje (un carácter: B para examinar o D para obtener de forma destructiva los mensajes).
3. El control de punto de sincronización (un carácter: S para que haya punto de sincronización o N para que no haya ningún punto de sincronización).
4. El nombre de la cola de destino (48 caracteres).

Si escribe incorrectamente alguno de estos parámetros, recibirá los correspondientes mensajes de error.

Para el ejemplo en COBOL, invoque el ejemplo de obtención en el entorno CICS especificando:

```
MVGT,9999,B,S,QUEUE.NAME
```

Para el ejemplo en C, invoque el ejemplo de obtención en el entorno CICS, especificando:

```
MCGT,9999,B,S,QUEUE.NAME
```

Cuando los mensajes se recuperan de la cola, se colocan en una cola de almacenamiento temporal CICS con el mismo nombre que la transacción CICS (por ejemplo, MCGT para el ejemplo en C).

A continuación se muestra un ejemplo de salida de los ejemplos de obtención:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE ***********************
```

Notas de uso

- Para mantener la simplicidad de los ejemplos, hay algunas diferencias funcionales menores entre las distintas versiones de lenguaje. Ninguna de las diferencias tiene que ver con la MQI.
- Si se especifica un nombre de cola que tenga más de 48 caracteres, su longitud se truncará al máximo de 48 caracteres, pero no se devolverá ningún mensaje de error.
- Antes de entrar en la transacción, pulse la tecla CLEAR.
- CSQ4CCJ1 solo puede mostrar correctamente mensajes de caracteres, porque solo visualiza hasta el primer carácter NULL (\0).
- En el campo numérico, especifique cualquier dígito comprendido en el rango de 1 a 9999. El valor que especifique tiene que ser un número positivo. Por ejemplo, para obtener un único mensaje, se pueden

especificar los valores 1, 01, 001 o 0001. Si se especifica un valor no numérico o negativo, es posible que reciba un error.

- Los mensajes de más de 24 526 bytes en C y 9 950 bytes en COBOL se truncan. Esto se debe a la forma en la que se utilizan las colas de almacenamiento temporal CICS.
- Para ambos programas, CSQ4CCK1 y CSQ4CVK1, especifique B en el parámetro get si desea examinar los mensajes; de lo contrario, especifique D. Esto realiza llamadas MQGET destructivas. Si especifica cualquier otro valor, recibirá un mensaje de error.
- En ambos programas, CSQ4CCJ1 y CSQ4CVJ1, especifique S en el parámetro de punto de sincronización para recuperar mensajes en dicho punto. Si especifica N en el parámetro de punto de sincronización, las llamadas MQGET se emitirán fuera de un punto de sincronización. Si especifica cualquier otro valor, recibirá un mensaje de error.

z/OS *Ejemplo de examen en z/OS*

El ejemplo de examen es una aplicación por lotes que muestra cómo examinar los mensajes de una cola utilizando la llamada MQGET.

La aplicación pasa por todos los mensajes de una cola, imprimiendo los primeros 80 bytes de cada uno de ellos. Puede utilizar esta aplicación para ver los mensajes de una cola sin cambiarlos.

Los programas fuente y el JCL de ejecución de ejemplo se proporcionan en los lenguajes COBOL, ensamblador, PL/I y C (consulte Tabla 173 en la página 1198).

Para iniciar la aplicación, edite y ejecute el JCL de ejecución de ejemplo, tal como se describe en “Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS” en la página 1197. Puede ver los mensajes en una de sus propias colas especificando el nombre de dicha cola en el JCL de ejecución.

Cuando se ejecuta la aplicación (y hay algunos mensajes en la cola), el conjunto de datos de salida se parece a esto:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

Si no hay mensajes en la cola, el conjunto de datos solo contiene los encabezamientos y el mensaje `Fin del informe`. Si se produce un error en cualquiera de las llamadas MQI, se añaden los códigos de terminación y de razón al conjunto de datos de salida.

z/OS *Diseño del ejemplo de examen en z/OS*

La aplicación de ejemplo de examen utiliza un único módulo de programa; se proporciona uno en cada uno de los lenguajes de programación soportados.

El flujo de la lógica del programa es:

1. Se abre un conjunto de datos de impresión y se imprime la línea de título del informe. Se comprueba que los nombres del gestor de colas y de la cola se hayan pasado desde el JCL de ejecución. Si se han pasado ambos nombres, se imprimen las líneas del informe que contienen los nombres. Si no, se imprime un mensaje de error, se cierra el conjunto de datos de impresión y se detiene el procesamiento.

La forma en la que el programa prueba los parámetros que se pasan del JCL depende del lenguaje en que está escrito el programa; para obtener más información, consulte “Consideraciones de diseño que dependen del lenguaje en z/OS” en la página 1213.

2. Se conecta con el gestor de colas utilizando la llamada MQCONN. Si esta llamada no es satisfactoria, se imprimen los códigos de terminación y de razón, se cierra el conjunto de datos de impresión y se detiene el procesamiento.
3. Se abre la cola usando la llamada MQOPEN con la opción MQOO_BROWSE. En la entrada de esta llamada, el programa usa el descriptor de conexión devuelto en el paso “2” en la página 1213. En la estructura de descriptor de objetos (MQOD), usa los valores predeterminados en todos los campos salvo el nombre de cola (que se pasó en el paso “1” en la página 1212). Si esta llamada no es satisfactoria, se imprimen los códigos de terminación y de razón, se cierra el conjunto de datos de impresión y se detiene el procesamiento.
4. Se examina el primer mensaje de la cola utilizando la llamada MQGET. En la entrada de esta llamada, el programa especifica:
   - La conexión y los descriptores de cola de los pasos “2” en la página 1213 y “3” en la página 1213
   - Una estructura MQMD con todos los campos establecidos a sus valores iniciales.
   - Dos opciones:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - Un búfer de 80 bytes para alojar los datos copiados del mensaje.

   La opción MQGMO_ACCEPT_TRUNCATED_MSG permite que la llamada se complete incluso si el mensaje es más largo que el búfer de 80 bytes especificado en la llamada. Si el mensaje es más largo que el búfer, el mensaje se trunca para que se ajuste al búfer y se establecen los códigos de terminación y de razón para dejar constancia de ello. El ejemplo se ha diseñado de modo que los mensajes se trunquen a 80 caracteres para que el informe sea fácil de leer. El tamaño del búfer se establece mediante una sentencia `DEFINE`, de modo que se puede cambiar fácilmente si así se desea.
5. Se realiza el siguiente bucle hasta que falle la llamada MQGET:
   a. Se imprime una línea del informe que muestra:
      - El número de secuencia del mensaje (es un recuento de las operaciones de examen).
      - La longitud real del mensaje (no la longitud truncada). Este valor se devuelve en el campo `DataLength` de la llamada MQGET.
      - Los primeros 80 bytes de los datos de mensaje.

   b. Se restablecen los campos `MsqId` y `CorrelId` de la estructura MQMD a nulos.
   c. Se examina el mensaje siguiente utilizando la llamada MQGET con estas dos opciones:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. Si la llamada MQGET falla, se comprueba el código de razón para ver si la llamada ha fallado porque el cursor para examinar ha llegado al final de la cola. En este caso, imprima el mensaje `End of report` (Fin de informe) y vaya al paso “7” en la página 1213 ; de lo contrario, imprima los códigos de terminación y razón, cierre el conjunto de datos de impresión y detenga el proceso.
7. Se cierra la cola con la llamada MQCLOSE empleando el descriptor de objeto devuelto en el paso “3” en la página 1213.
8. Se desconecta del gestor de colas con la llamada MQDISC empleando el descriptor de conexión devuelto en el paso “2” en la página 1213.
9. Se cierra el conjunto de datos de impresión y se detiene el procesamiento.

z/OS *Consideraciones de diseño que dependen del lenguaje en z/OS*

Se proporcionan módulos fuente para el ejemplo de examen en cuatro lenguajes de programación.

Existen dos diferencias principales entre los módulos fuente:

- Al probar los parámetros pasados desde el JCL de ejecución, los módulos de lenguaje COBOL, PL/I y ensamblador buscan el carácter de coma (,). Si el JCL pasa `PARM=(,LOCALQ1)`, la aplicación intenta abrir la cola LOCALQ1 en el gestor de colas predeterminado. Si no hay ningún nombre después de la coma (o ninguna coma), la aplicación devuelve un error. El módulo C no busca el carácter de coma. Si el JCL pasa un único parámetro (por ejemplo, `PARM=('LOCALQ1')`), el módulo C lo usa como nombre de cola en el gestor de colas predeterminado.
- Para mantener el módulo de ensamblador simple, utiliza el formato de fecha *aa/ddd* (por ejemplo, 05/116) cuando crea el informe de impresión. Los otros módulos utilizan la fecha de calendario en formato *mm/dd/aa*.

z/OS *Ejemplo Imprimir mensaje (Print Message) en z/OS*

El ejemplo Imprimir mensaje es una aplicación por lotes que muestra cómo eliminar todos los mensajes de una cola utilizando la llamada MQGET.

El ejemplo Imprimir mensaje utiliza tres parámetros:

1. El nombre del gestor de colas
2. El nombre de la cola de origen
3. Un parámetro opcional para las propiedades

También imprime, para cada mensaje, los campos del descriptor de mensaje, seguidos por los datos del mensaje. El programa imprime los datos tanto en hexadecimal como en formato de caracteres (si son imprimibles). Si un carácter no es imprimible, el programa lo sustituye por un carácter de punto (.). Puede utilizar el programa al diagnosticar problemas con una aplicación que está colocando mensajes en una cola.

Los valores permitidos para el parámetro de propiedades son:

Tabla 185. Valores permitidos para el parámetro de propiedad

Valor	Comportamiento
0	Comportamiento predeterminado. Las propiedades que se entregan a la aplicación dependen del atributo de cola **`PropertyControl`** del que se recupera el mensaje.

1	Se crea un manejador de mensajes y se utiliza con MQGET. Las propiedades del mensaje, excepto las contenidas en el descriptor de mensaje (o extensión), se visualizan como en el descriptor de mensaje. Por ejemplo:
	`****Message properties****`
	`  property name: property value`
	O bien, si no hay propiedades disponibles:
	`****Message properties****`
	`  None`
	Los valores numéricos se visualizan utilizando `printf`, los valores de serie se escriben entre comillas simples y las series de bytes se escriben entre X y comillas simples, al igual que en el descriptor de mensaje.

2	Se ha especificado MQGMO_NO_PROPERTIES, por lo que solo se devolverán las propiedades del descriptor de mensaje.
3	Se ha especificado MQGMO_PROPERTIES_FORCE_MQRFH2, por lo que se devuelven todas las propiedades en los datos del mensaje.

Tabla 185. Valores permitidos para el parámetro de propiedad (continuación)	
Valor	**Comportamiento**
4	Se especifica MQGMO_PROPERTIES_COMPATIBILITY, de modo que se pueden devolver todas las propiedades en función de si se incluye una propiedad IBM MQ ; de lo contrario, se descartan las propiedades.

Puede cambiar la aplicación para que examine los mensajes, en lugar de eliminarlos de la cola. Para hacer esto, compile con la opción -DBROWSE, para definir la macro BROWSE, tal como se indica en "Diseño del ejemplo de mensaje de impresión en z/OS" en la página 1216. Se proporciona el código ejecutable en la biblioteca SCSQLOAD. El módulo CSQ4BCG0 está compilado con -DBROWSE; el módulo CSQ4BCG1 lee la cola de manera destructiva.

La aplicación tiene un único programa fuente, que está escrito en lenguaje C. También se proporciona el código JCL de ejecución de ejemplo (consulte Tabla 174 en la página 1199).

Para iniciar la aplicación, edite y ejecute el JCL de ejecución de ejemplo, tal como se describe en "Preparación y ejecución de aplicaciones de ejemplo para el entorno de proceso por lotes de z/OS" en la página 1197. Cuando ejecute la aplicación (y haya algunos mensajes en la cola), los datos de salida tendrán un aspecto similar al de Figura 139 en la página 1216.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR  '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '  '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figura 139. Ejemplo de un informe de la aplicación de ejemplo Imprimir mensaje

z/OS *Diseño del ejemplo de mensaje de impresión en z/OS*

La aplicación de ejemplo de mensaje de impresión utiliza un solo programa escrito en C.

El flujo de la lógica del programa es:

1. Se comprueba que los nombres del gestor de colas y de la cola se hayan pasado desde el JCL de ejecución. Si no se han pasado, se imprime un mensaje de error y se detiene el procesamiento.
2. Se conecta con el gestor de colas utilizando la llamada MQCONN. Si esta llamada no se realiza correctamente, se imprimen los códigos de terminación y de razón, y se detiene el procesamiento; de lo contrario, se imprime el nombre del gestor de colas.
3. Se abre la cola usando la llamada MQOPEN con la opción MQOO_INPUT_SHARED.

   Nota: Si desea que la aplicación examine los mensajes en lugar de eliminarlos de la cola, compile el ejemplo con -DBROWSE o añada `#define BROWSE` al principio del código fuente. Cuando se hace esto, el preprocesador de macros añade la línea en el programa que selecciona la opción MQOO_BROWSE en la compilación.

En la entrada de esta llamada, el programa usa el descriptor de conexión devuelto en el paso “2” en la página 1216. En la estructura de descriptor de objetos (MQOD), usa los valores predeterminados en todos los campos salvo el nombre de cola (que se pasó en el paso “1” en la página 1216). Si esta llamada no se realiza correctamente, se imprimen los códigos de terminación y de razón, y se detiene el procesamiento; de lo contrario, se imprime el nombre de la cola.

4. Si se usa un manejador de mensajes para obtener las propiedades de mensaje, se usa MQCRTMH para crear ese manejador para utilizarlo con las llamadas MQGET posteriores. Si esta llamada no es satisfactoria, se imprimen los códigos de terminación y de razón, y se detiene el proceso.
5. Se definen las opciones del mensaje de obtención para reflejar la acción de petición de cualquier propiedad de mensaje.
6. Se realiza el siguiente bucle hasta que falle la llamada MQGET:
   a. Se inicializa el búfer a espacios en blanco para que los datos de mensaje no se corrompan con datos que ya estuvieran en el búfer.
   b. Se establecen los campos `MsgId` y `CorrelId` de la estructura MQMD a nulos para que la llamada MQGET seleccione el primer mensaje de la cola.
   c. Se obtiene un mensaje de la cola con la llamada MQGET. En la entrada de esta llamada, el programa especifica:
      - La conexión y los descriptores de objeto de los pasos “2” en la página 1216 y “3” en la página 1216.
      - Una estructura MQMD con todos los campos establecidos en sus valores iniciales. (`MsgId` y `CorrelId` se restablecen en nulos para cada llamada MQGET.)
      - La opción MQGMO_NO_WAIT.

        Nota: Si desea que la aplicación examine los mensajes en lugar de eliminarlos de la cola, compile el ejemplo con -DBROWSE o añada `#define BROWSE` al principio del código fuente. Cuando se hace esto, el preprocesador de macros añade la línea en el programa que selecciona la opción MQGMO_BROWSE_NEXT a la compilación. Cuando se utiliza esta opción en una llamada contra una cola en la que no se ha utilizado previamente ningún cursor para examinar con el descriptor de objeto actual, el cursor para examinar se coloca lógicamente delante del primer mensaje.
      - Un búfer de 64 KB para contener los datos copiados del mensaje.
   d. Se invoca la subrutina printMD. Esto imprime el nombre de cada campo en el descriptor de mensaje, seguido de su contenido.
   e. Si se ha creado un descriptor de mensaje en el paso “4” en la página 1217, se invoca la subrutina printProperties para visualizar las propiedades de menaje que haya.
   f. Se imprime la longitud del mensaje, seguida de los datos del mensaje. Cada línea de datos de mensaje tiene este formato:
      - Posición relativa (en hexadecimal) de esta parte de los datos.
      - 16 bytes de datos hexadecimales.
      - Los mismos 16 bytes de datos en formato de caracteres, si son imprimibles (los caracteres no imprimibles se sustituyen por puntos).
7. Si la llamada MQGET falla, se prueba el código de razón para ver si ha fallado porque ya no hay más mensajes en la cola. En este caso, se imprime el mensaje: 'No more messages'; de lo contrario, imprimen los códigos de terminación y de razón. En ambos casos, vaya al paso “9” en la página 1218.

   Nota: La llamada MQGET falla si encuentra un mensaje con más de 64 KB de datos. Para cambiar el programa de forma que maneje mensajes más grandes, se puede realizar una de las acciones siguientes:

   - Añadir la opción MQGMO_ACCEPT_TRUNCATED_MSG a la llamada MQGET para que la llamada obtenga los primeros 64 KB de datos y descarte el resto.
   - Hacer que el programa deje el mensaje en la cola cuando se encuentre uno con esta cantidad de datos.

- Incrementar el tamaño del búfer.

8. Si ha creado un descriptor de mensajes en el paso “4” en la página 1217, llame a MQDLTMH para suprimirlo.
9. Se cierra la cola con la llamada MQCLOSE empleando el descriptor de objeto devuelto en el paso “3” en la página 1216.
10. Se desconecta del gestor de colas con la llamada MQDISC empleando el descriptor de conexión devuelto en el paso “2” en la página 1216.

z/OS *Ejemplo de atributos de cola en z/OS*

El ejemplo de atributos de cola es una aplicación CICS de modalidad conversacional que ilustra el uso de las llamadas MQINQ y MQSET.

Muestra cómo consultar los valores de los atributos **InhibitPut** e **InhibitGet** de las colas y cómo cambiarlos para que los programas no puedan colocar mensajes en una cola ni obtener mensajes de ella. Puede que le interese *bloquear* una cola de esta forma cuando esté probando un programa.

Para evitar interferencias accidentales con sus propias colas, este ejemplo solo trabaja en un objeto de cola que tenga los caracteres CSQ4SAMP en los primeros ocho bytes de su nombre. Sin embargo, el código fuente incluye comentarios para mostrarle cómo eliminar esta restricción.

Los fuentes de los programas se suministran en COBOL, ensamblador y C (consulte Tabla 180 en la página 1203).

La versión en lenguaje ensamblador del ejemplo utiliza código reentrante. Para ello, observará que el código de cada llamada MQI en dicha versión del ejemplo incluye la palabra clave MF; por ejemplo:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(La palabra clave VL significa que se puede utilizar la transacción proporcionada por CICS Execution Diagnostic Facility (CEDF) para la depuración del programa). Para obtener más información sobre cómo escribir programas reentrantes, consulte Desarrollo en ensamblador en System/390.

Para iniciar la aplicación, inicie el sistema CICS y utilice las transacciones CICS siguientes:

- En COBOL, MVC1
- En ensamblador, MAC1
- En C, MCC1

Puede cambiar el nombre de cualquiera de estas transacciones modificando el CSD mencionado en el paso 3.

Diseño del ejemplo

Cuando se inicia el ejemplo, aparece una correlación por pantalla que tiene campos para:

- Nombre de la cola
- La petición del usuario (las acciones válidas son: inquire, allow, or inhibit) (interrogar, permitir o inhibir respectivamente).
- Estado actual de las operaciones de colocación de la cola.
- Estado actual de las operaciones de obtención de la cola.

Los dos primeros campos se usan para la entrada del usuario. Los dos últimos campos los rellena la aplicación: muestran la palabra `INHIBITED` (inhibida) o la palabra `ALLOWED` (permitida).

La aplicación valida los valores que se especifican en los dos primeros campos. Comprueba que el nombre de cola empieza por los caracteres CSQ4SAMP y que ha especificado una de las tres peticiones válidas en el campo Acción. La aplicación pasa toda la entrada a mayúsculas, por lo que no se puede utilizar ninguna cola que tenga un nombre con caracteres en minúscula.

Si se especifica `inquire` en el campo **Acción**, el flujo a través de la lógica del programa es:

1. Se abre la cola usando la llamada MQOPEN con la opción MQOO_INQUIRE.
2. Se invoca MQINQ con los selectores MQIA_INHIBIT_GET y MQIA_INHIBIT_PUT.
3. Se cierra la cola con la llamada MQCLOSE.
4. Se analizan los atributos devueltos en el parámetro **IntAttrs** de la llamada MQINQ y se mueven las palabras INHIBITED o ALLOWED, según corresponda, a los campos de pantalla relevantes.

Si se especifica `inhibit` en el campo **Acción**, el flujo a través de la lógica del programa es:

1. Se abre la cola usando la llamada MQOPEN con la opción MQOO_SET.
2. Se invoca MQSET con los selectores MQIA_INHIBIT_GET y MQIA_INHIBIT_PUT, y con los valores MQQA_GET_INHIBITED y MQQA_PUT_INHIBITED en el parámetro **IntAttrs**.
3. Se cierra la cola con la llamada MQCLOSE.
4. Se mueve la palabra INHIBITED a los correspondientes campos de pantalla.

Si se especifica `allow` en el campo **Acción**, la aplicación realiza un procesamiento similar al de la petición inhibit. Las únicas diferencias son los valores de los atributos y las palabras que aparecen por pantalla.

Cuando la aplicación abre la cola, utiliza el descriptor de conexión predeterminado con el gestor de colas. (CICS establece una conexión con el gestor de colas al iniciar el sistema CICS .) La aplicación puede capturar los errores siguientes en esta etapa:

- La aplicación no está conectada con el gestor de colas.
- La cola no existe.
- El usuario no tiene autorización para acceder a la cola.
- La aplicación no tiene autorización para abrir a la cola.

En el caso de otros errores de MQI, la aplicación muestra los códigos de terminación y de razón.

z/OS *Ejemplo de gestor de correo en z/OS*

La aplicación de ejemplo de gestor de correo es una suite de programas que ilustra el envío y la recepción de mensajes, tanto dentro de un entorno único como en distintos entornos. La aplicación es un sistema de correo electrónico sencillo que permite a los usuarios intercambiar mensajes, incluso si utilizan gestores de colas diferentes.

La aplicación ilustra cómo crear colas utilizando la llamada MQOPEN y colocando mandato sIBM MQ for z/OS en la cola de entrada de mandatos del sistema.

Se proporcionan tres versiones de la aplicación:

- Una aplicación CICS escrita en COBOL
- Una aplicación TSO escrita en COBOL.
- Una aplicación TSO escrita en C.

z/OS *Preparación del ejemplo de gestor de correo en z/OS*

El gestor de correo se proporciona en versiones que ejecutan en dos entornos. Los preparativos que hay que llevar a cabo antes de ejecutar la aplicación dependerán del entorno que desee utilizar.

Los usuarios pueden acceder a las colas de correo y de apodo desde TSO y CICS siempre que los ID de usuario de inicio de sesión sean los mismos en cada sistema.

Para poder enviar mensajes a otro gestor de colas, hay que configurar un canal de mensajes a dicho gestor de colas. Para ello, use la función de control de canales de IBM MQ, descrita en Función de control de canales.

Preparación del ejemplo para el entorno TSO

Siga estos pasos:

1. Prepare el ejemplo tal como se describe en “Preparación de aplicaciones de ejemplo para el entorno TSO en z/OS” en la página 1200.
2. Adapte la CLIST proporcionada para el ejemplo a fin de definir:
   - La ubicación de los paneles.
   - La ubicación del archivo de mensaje.
   - La ubicación de los módulos de carga.
   - El nombre del gestor de colas que desee utilizar con la aplicación.

   Se proporciona una CLIST separada por cada lenguaje del ejemplo:
   - Para la versión en COBOL: CSQ4RVD1.
   - Para la versión en C: CSQ4RCD1.
3. Asegúrese de que las colas utilizadas por la aplicación estén disponibles en el gestor de colas. (Las colas se definen en CSQ4CVD).

Nota: VS COBOL II no soporta multitarea con ISPF. Esto significa que no se puede utilizar la aplicación de ejemplo de gestión de correo en ambos lados de una pantalla dividida. Si lo hace, los resultados serán impredecibles.

z/OS *Ejecución del ejemplo de gestión de correo en z/OS*

Para iniciar el ejemplo en el entorno de CICS Transaction Server para z/OS, ejecute la transacción MAIL. Si todavía no iniciado sesión en CICS, la aplicación le solicita que entre un ID de usuario al que le puede enviar correo.

Al iniciarse, la aplicación, abre la cola de correo. Si esta cola no existe, la aplicación creará una. Las colas de correo tienen nombres con el formato CSQ4SAMP.MAILMGR. *userid*, donde *userid* depende del entorno:

En TSO
ID de TSO del usuario

En CICS
El inicio de sesión en CICS o el ID de usuario que especifica el usuario cuando se le solicita al iniciar el gestor de correo

Todas las partes de los nombres de cola que utiliza el gestor de correo tienen que estar en mayúscula.

A continuación, la aplicación presenta un panel de menús que tiene opciones para:

- Leer el correo entrante.
- Enviar correo.
- Crear un apodo.

El panel de menús también muestra cuántos mensajes están esperando en la cola de correo. Cada una de las opciones de menú muestra un panel adicional:

Leer el correo entrante.
El gestor de correo muestra la lista de mensajes que están en la cola de correo. (Solo se muestran los primeros 99 mensajes de la cola). Para ver un ejemplo de este panel, consulte Figura 142 en la página 1224. Cuando se selecciona un mensaje en esta lista, se visualiza el contenido del mensaje (consulte Figura 143 en la página 1225).

Enviar correo.
Un panel solicita que se especifique:
- El nombre del usuario al que se desea enviar un mensaje.
- El nombre del gestor de colas que es propietario de la cola de correo.
- El texto del mensaje.

En el campo de nombre de usuario, se puede especificar un ID de usuario o un apodo creados con el gestor de correo. El campo de nombre del gestor de colas se puede dejar en blanco si la cola de correo del usuario es propiedad del mismo gestor de colas que se está utilizando, y hay que dejarlo en blanco si se ha especificado un apodo en el campo de nombre de usuario:

- Si solo se especifica un nombre de usuario, el programa asumirá en primer lugar que el nombre es un apodo y enviará el mensaje al objeto definido por ese nombre. Si no existe tal apodo, el programa intentará enviar el mensaje a una cola local que tenga ese nombre.
- Se se especifican un nombre de usuario y un nombre de gestor de colas, el programa enviará a la cola de correo definida por esos dos nombres.

Por ejemplo, si se desea enviar un mensaje al usuario JONESM en el gestor de colas remoto QM12, se le puede enviar un mensaje de dos maneras:

- Usando ambos campos para especificar el usuario JONESM en el gestor de colas QM12.
- Definiendo un apodo (por ejemplo, MARY) para dicho usuario y enviándole un mensaje poniendo MARY en el campo del nombre de usuario y nada en el campo del nombre de gestor de colas.

Crear un apodo.
Se puede definir un nombre fácil de recordar que se puede utilizar al enviar un mensaje a otro usuario con el que se contacte con frecuencia. Se le solicitará que especifique el ID del otro usuario y el nombre del gestor de colas que es propietario de la cola de correo.

Los apodos son colas que tienen nombres con el formato CSQ4SAMP.MAILMGR. *userid.nickname*, donde *ID_usuario* es su propio ID de usuario y *apodo* es el apodo que desea utilizar. Con los nombres estructurados de esta forma, cada usuario puede tener su propio conjunto de apodos.

El tipo de cola que crea el programa dependerá de cómo se completen los campos del panel Crear un apodo:

- Si solo se especifica un nombre de usuario, o el nombre del gestor de colas es el mismo que el del gestor de colas con el que está conectado el gestor de correo, el programa creará una cola de alias.
- Si se especifican un nombre de usuario y un nombre de gestor de colas (y el gestor de colas no es el que está conectado con el gestor de correo), el programa creará una definición local de una cola remota. El programa no comprueba la existencia de la cola a la que se resuelve esta definición, ni siquiera si existe el gestor de colas remoto.

Por ejemplo, si su propio ID de usuario es SMITHK y se crea el apodo MARY para el usuario JONESM (que usa el gestor de colas remoto QM12), el programa de apodo creará una definición local de una cola remota llamada CSQ4SAMP.MAILMGR.SMITHK.MARY. Esta definición resuelve a la cola de correo de Mary, que es CSQ4SAMP.MAILMGR.JONESM en el gestor de colas QM12. Si utiliza el gestor de colas QM12 usted mismo, el programa crea en su lugar una cola de alias con el mismo nombre (CSQ4SAMP.MAILMGR.SMITHK.MARY).

La versión en C de la aplicación TSO hace un mayor uso de las prestaciones de tratamiento de mensajes de ISPF que la versión en COBOL. Observará que los mensajes de error visualizados de la versión en C son distintos de los visualizados en la versión en COBOL.

z/OS *Diseño del ejemplo de gestor de correo en z/OS*

En las secciones siguientes se describen cada uno de los programas que componen la aplicación de ejemplo de gestión de correo.

Las relaciones entre los programas y los paneles que utiliza la aplicación se muestra en Figura 140 en la página 1222 para la versión TSO y Figura 141 en la página 1223 para la versión de CICS Transaction Server for z/OS.

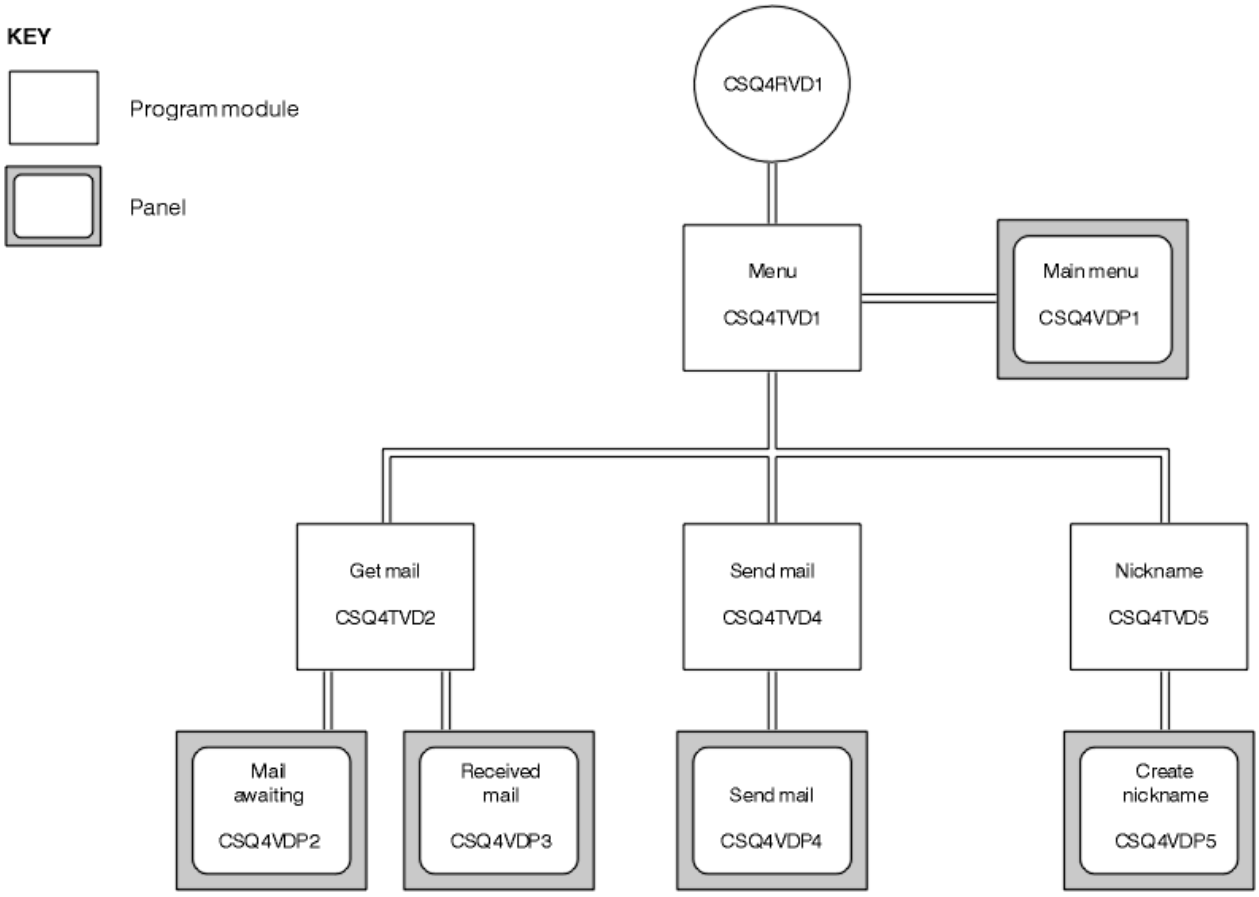


Figura 140. Programas y paneles para las versiones TSO del gestor de correo

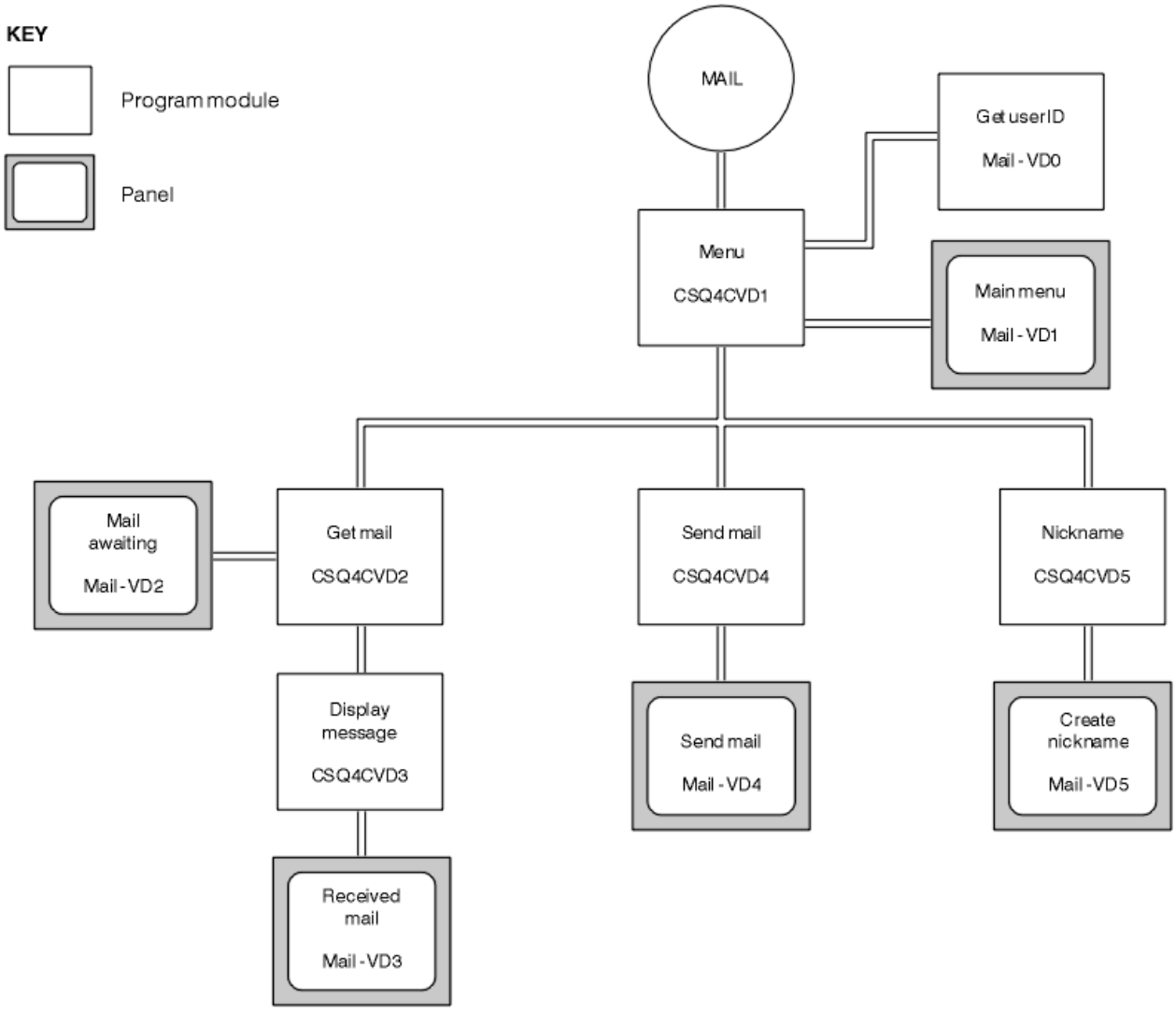


Figura 141. Programas y paneles para la versión de CICS del gestor de correo

z/OS *Programa de menú en z/OS*

En el entorno TSO, el programa de menú lo invoca la CLIST. En el entorno CICS, la transacción MAIL invoca el programa.

El programa de menú (CSQ4TVD1 para TSO, CSQ4CVD1 para CICS) es el programa inicial de la suite. Muestra el menú (CSQ4VDP1 para TSO, VD1 para CICS) e invoca los otros programas cuando se seleccionan en el menú.

El programa obtiene en primer lugar el ID del usuario:

- En la versión CICS del programa, si el usuario ha iniciado sesión en CICS, el ID de usuario se obtiene utilizando el mandato CICS ASSIGN USERID. Si el usuario no ha iniciado sesión, el programa muestra el panel de inicio de sesión (CSQ4VD0) para solicitarle al usuario un identificador de usuario. No hay ningún proceso de seguridad en este programa; el usuario puede dar cualquier ID de usuario.
- En la versión TSO, el ID del usuario se obtiene de TSO en la CLIST. Se pasa al programa de menú como una variable en la agrupación compartida del ISPF.

Una vez que el programa ha obtenido el ID de usuario, comprueba que el usuario tiene una cola de correo (CSQ4SAMP.MAILMGR. *ID_usuario*). Si no existe una cola de correo, el programa crea una colocando un mensaje en la cola de entrada de comandos del sistema. El mensaje contiene el mandato IBM MQ for z/OS DEFINE QLOCAL. La definición de objeto que usada por este comando establece la profundidad máxima de la cola a 9999 mensajes.

El programa también crea una cola dinámica temporal para manejar las respuestas de la cola de entrada de comandos del sistema. Para ello, utiliza la llamada MQOPEN, especificando SYSTEM.DEFAULT.MODEL.QUEUE como plantilla de la cola dinámica. El gestor de colas crea la cola dinámica temporal con un nombre que tiene el prefijo CSQ4SAMP; el resto del nombre lo genera el gestor de colas.

A continuación, el programa abre la cola de correo del usuario y encuentra el número de mensajes en la cola consultando la profundidad actual de la cola. Para hacer esto, el programa utiliza la llamada MQINQ especificando el selector MQIA_CURRENT_Q_DEPTH.

A continuación, el programa realiza un bucle que muestra el menú y procesa la selección que hace el usuario. El bucle se para cuando el usuario pulsa la tecla PF3. Cuando se realiza una selección válida, se inicia el correspondiente programa; en caso contrario, se muestra un mensaje de error.

z/OS *Programas de obtención de correo y de visualización de mensajes en z/OS*

En las versiones TSO de la aplicación, las funciones get-mail (obtener correo) y display-message (visualizar mensaje) las realiza el mismo programa (CSQ4TVD2). En la versión CICS de la aplicación, estas funciones se realizan mediante programas independientes (CSQ4CVD2 y CSQ4CVD3).

El panel de espera de correo (CSQ4VDP2 para TSO, VD2 para CICS; consulte Figura 142 en la página 1224 para ver un ejemplo) muestra todos los mensajes que están en la cola de correo del usuario. Para crear esta lista, el programa utiliza la llamada MQGET para examinar todos los mensajes de la cola, guardando información sobre cada uno de ellos. Además de la información mostrada, el programa registra los `MsgId` y `CorrelId` de cada mensaje.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figura 142. Ejemplo de un panel que muestra una lista de mensajes en espera

Desde el panel de espera de correo, el usuario puede seleccionar un mensaje y mostrar el contenido del mensaje (consulte Figura 143 en la página 1225 para ver un ejemplo). El programa utiliza la llamada MQGET para eliminar este mensaje de la cola, utilizando el `MsgId` y el `CorrelId` que el programa registró al examinar todos los mensajes. Esta llamada MQGET se realiza con la opción MQGMO_SYNCPOINT. El programa muestra el contenido del mensaje y luego declara un punto de sincronización: esto confirma la llamada MQGET, con lo que el mensaje deja de existir.

```
-------------------- IBM MQ for z/OS Sample Programs --------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
------------------------------------ Message -------------------------------
| HELLO FROM JOHNJ                                |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
'-----------------------------------------------------------------------------'
```

Figura 143. Ejemplo de un panel que muestra el contenido de un mensaje

Una extensión obvia a la función proporcionada por el gestor de correo consiste en dar al usuario la opción de dejar el mensaje en la cola tras consultar su contenido. Para ello, hay que restituir la llamada MQGET que elimina el mensaje de la cola tras ser visualizado.

z/OS *Programa de envío de correo en z/OS*

Cuando el usuario haya completado el panel de envío de correo (CSQ4VDP4 para TSO, VD4 para CICS), el programa de envío de correo (CSQ4TVD4 para TSO, CSQ4CVD4 para CICS) coloca el mensaje en la cola de correo del destinatario.

Para ello, el programa utiliza la llamada MQPUT1. El destino del mensaje depende de cómo haya rellenado el usuario los campos en el panel de envío de correo:

- Si el usuario solo ha especificado un nombre de usuario, el programa asumirá en primer lugar que el nombre es un apodo y enviará el mensaje al objeto definido por ese nombre. Si no existe tal apodo, el programa intentará enviar el mensaje a una cola local que tenga ese nombre.
- Si el usuario ha especificado un nombre de usuario y un nombre de gestor de colas, el programa enviará a la cola de correo definida por esos dos nombres.

El programa no acepta mensajes en blanco y elimina los espacios en blanco iniciales de cada línea del mensaje de texto.

Si la llamada MQPUT1 se ha realizado correctamente, el programa muestra un mensaje en el que aparecen los nombres de usuario y del gestor de colas el que se ha colocado el mensaje. Si la llamada no es satisfactoria, el programa comprueba específicamente los códigos de razón que indican que la cola o el gestor de colas no existen; estos son MQRC_UNKNOWN_OBJECT_NAME y MQRC_UNKNOWN_OBJECT_Q_MGR. El programa muestra su propio mensaje de error en cada uno de estos errores; en los demás errores, el programa muestra los códigos de terminación y razón devueltos por la llamada.

z/OS *Programa de apodo en z/OS*

Cuando el usuario define un apodo, el programa de apodo (CSQ4TVD5 para TSO, CSQ4CVD5 para CICS) crea una cola que tiene el apodo como parte de su nombre.

El programa hace esto colocando un mensaje en la cola de entrada del comandos del sistema. El mensaje contiene el mandato de IBM MQ for z/OS DEFINE QALIAS o DEFINE QREMOTE. El tipo de cola que crea el programa depende de cómo haya rellenado el usuario los campos del panel Crear apodo (CSQ4VDP5 para TSO, VD5 paraCICS):

- Si el usuario solo ha especificado un nombre de usuario, o el nombre del gestor de colas es el mismo que el del gestor de colas con el que está conectado el gestor de correo, el programa creará una cola de alias.
- Si el usuario ha especificado un nombre de usuario y un nombre de gestor de colas (y el gestor de colas no es el que está conectado con el gestor de correo), el programa creará una definición local de una cola remota. El programa no comprueba la existencia de la cola a la que se resuelve esta definición, ni siquiera si existe el gestor de colas remoto.

El programa también crea una cola dinámica temporal para manejar las respuestas de la cola de entrada de comandos del sistema.

Si el gestor de colas no puede crear la cola de apodo por una razón que el programa contempla (por ejemplo, la cola ya existe), el programa muestra su propio mensaje de error. Si el gestor de colas no puede crear la cola por una razón que el programa no contempla, este mostrará hasta dos de los mensajes de error que le devuelva el servidor de comandos.

Nota: Por cada apodo, el programa de apodo solo crea una cola alias o una definición local de una cola remota. Las colas locales a las que resuelven estos nombres de cola solo se crean cuando se usa el ID de usuario contenido en el apodo para iniciar la aplicación de gestión de correo.

z/OS *Ejemplo de comprobación de crédito en z/OS*

La aplicación de ejemplo de comprobación de crédito es una suite de programas que ilustra cómo utilizar muchas de las características proporcionadas por IBM MQ for z/OS. Muestra cómo muchos de los programas de componente de una aplicación pueden intercambiar mensajes entre sí mediante técnicas de encolado de mensajes.

El ejemplo se puede ejecutar como una aplicación CICS autónoma. No obstante, para mostrar cómo se diseña una aplicación de puesta en cola de mensajes que utiliza los recursos que proporcionan los entornos CICS e IMS, también se proporciona un módulo como un programa de proceso de mensajes por lotes de IMS. Esta extensión del ejemplo se describe en "Extensión IMS del ejemplo de comprobación de crédito en z/OS" en la página 1237.

También se puede ejecutar el ejemplo en más de un gestor de colas y enviar mensajes entre cada instancia de la aplicación. Para ello, consulte "Ejemplo de comprobación de crédito con varios gestores de colas en z/OS" en la página 1237.

Los programas CICS se entregan en C y COBOL. El único programa IMS sólo se entrega en C. Los conjuntos de datos proporcionados se muestran en Tabla 182 en la página 1204 y Tabla 184 en la página 1206.

La aplicación ilustra un método de evaluación de riesgos cuando los clientes bancarios piden préstamos. La aplicación muestra cómo podría trabajar un banco para procesar las solicitudes de préstamo de dos maneras:

- Al tratar directamente con un cliente, el personal del banco desea un acceso inmediato a la información de cuenta y riesgo crediticio.
- Al usar aplicaciones desarrolladas, el personal bancario puede presentar una serie de solicitudes de información de riesgo crediticio y de cuentas, y ocuparse de las respuestas en un momento posterior.

Los detalles financieros y de seguridad de la aplicación se han mantenido simples para que las técnicas de encolamiento de mensajes queden claras.

z/OS *Preparación y ejecución del programa de ejemplo Credit Check (Comprobación de crédito) en z/OS*

Para preparar y ejecutar el ejemplo Credit Check, realice los pasos siguientes:

1. Cree el conjunto de datos VSAM que contiene información acerca de algunas cuentas de ejemplo. Realice esta tarea editando y ejecutando el JCL proporcionado en el conjunto de datos CSQ4FILE.
2. Realice los pasos de "Preparación de las aplicaciones de ejemplo para el entorno CICS en z/OS" en la página 1202. (Los pasos adicionales que debe realizar si quiere utilizar la extensión IMS para el

ejemplo, se describen en “Extensión IMS del ejemplo de comprobación de crédito en z/OS” en la página 1237.)

3. Inicie el supervisor desencadenante de CKTI (proporcionado con IBM MQ for z/OS) en la cola CSQ4SAMP.INITIATION.QUEUE, utilizando la transacción CKQC de CICS.
4. Para iniciar la aplicación, inicie el sistema CICS y utilice la transacción MVB1.
5. Seleccione la consulta **Immediate** (inmediata) o **Batch** (lote) en el primer panel.

   Los paneles de consulta inmediata y por lotes son similares; Figura 144 en la página 1227 muestra el panel Consulta inmediata.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . _______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : ____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figura 144. Panel Consulta inmediata para la aplicación de ejemplo Credit Check

6. Especifique un número de cuenta y un importe de préstamo en los campos correspondientes. Consulte “Entrada de información en los paneles de consulta” en la página 1227 para obtener instrucciones sobre la información que se debe especificar en estos campos.

Entrada de información en los paneles de consulta

La aplicación de ejemplo Credit Check comprueba que los datos que especifique en el campo **Importe solicitado** de los paneles de consulta se encuentran en forma de enteros.

Si especifica uno de los números de cuenta siguientes, la aplicación busca el nombre de cuenta adecuado, el saldo medio de la cuenta y el índice de solvencia de crédito en el conjunto de datos CSQ4BAQ de VSAM:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

Puede especificar cualquier información en los otros campos, o dejarlos vacíos. La aplicación mantiene la información que especifique y devuelve la misma información en los informes que genera.

z/OS *Diseño del ejemplo de comprobación de crédito en z/OS*

En esta sección se describe el diseño de cada uno de los programas de que consta la aplicación de ejemplo de comprobación de crédito.

Para obtener más información sobre algunas de las técnicas que se han tenido en cuenta durante el diseño de la aplicación, consulte “Consideraciones de diseño para el ejemplo de comprobación de crédito en z/OS” en la página 1234.

Figura 145 en la página 1229 muestra los programas que conforman la aplicación y, también, las colas a las que prestan servicio estos programas. En esta figura, el prefijo CSQ4SAMP se ha omitido de todos los nombres de cola para que la figura sea más comprensible.

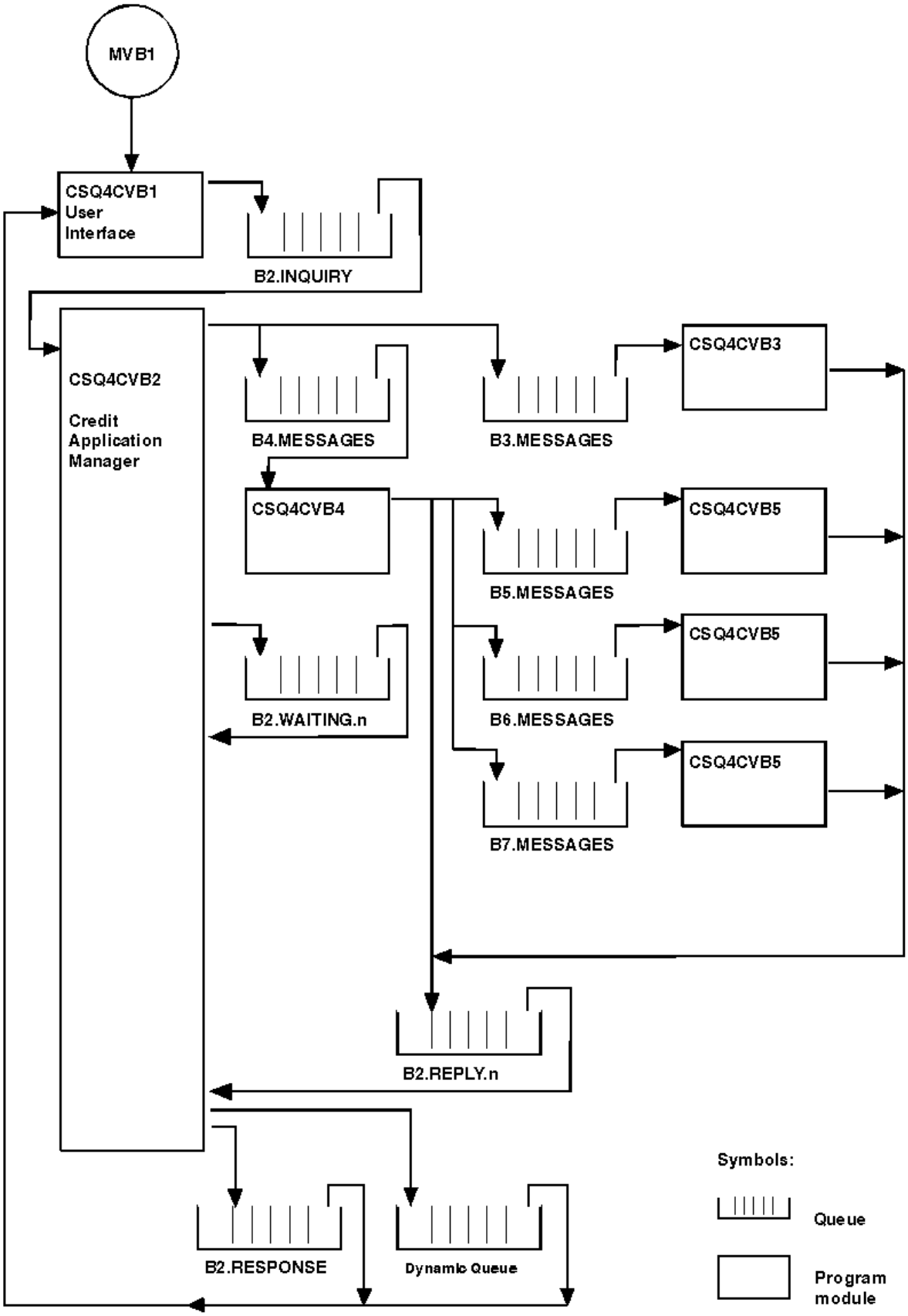


Figura 145. Programas y colas de la aplicación de ejemplo de comprobación de crédito (solo programas COBOL)

z/OS *Programa de interfaz de usuario (CSQ4CVB1) en z/OS*

Cuando se inicia la transacción CICS MVB1 en modo conversacional, se inicia la interfaz de usuario de la aplicación.

Este programa coloca los mensajes de consulta en la cola CSQ4SAMP.B2.INQUIRY y obtiene respuestas a esas consultas de una cola de respuestas que se especifica al realizar la consulta. Desde la interfaz de usuario se pueden enviar consultas inmediatas o por lotes:

- En el caso de las consultas inmediatas, el programa crea una cola dinámica temporal que utiliza como cola de respuesta. Esto significa que cada consulta tiene su propia cola de respuestas.
- En el caso de las consultas por lotes, el programa de interfaz de usuario obtiene respuestas de la cola CSQ4SAMP.B2.RESPONSE. A efectos de simplicidad, el programa obtiene las respuestas a todas sus consultas de esta cola de respuestas. Es fácil ver que un banco podría querer utilizar una cola de respuestas aparte por cada usuario de MVB1, de modo que cada uno de ellos solo pueda ver las respuestas a las consultas iniciadas por el mismo.

Las siguientes son diferencias importantes entre las propiedades de los mensajes usados en la aplicación en modo inmediato y por lotes:

- En el funcionamiento por lotes, los mensajes tienen una prioridad baja, de modo que se procesan una vez especificadas las solicitudes de préstamo en modo inmediato. Además, los mensajes son persistentes, por lo que se recuperarán si la aplicación o el gestor de colas tienen que reiniciarse.
- En el funcionamiento inmediato, los mensajes tienen una prioridad alta, de modo que se procesan antes de que se especifiquen las solicitudes de préstamo en modo por lotes. Además, los mensajes no son persistentes, por lo que se descartarán si la aplicación o el gestor de colas tienen que reiniciarse.

Sin embargo, en todos los casos, las propiedades de los mensajes de solicitud de préstamo se propagan a través de la aplicación. Así pues, por ejemplo, todos los mensajes que resultantes de una solicitud de prioridad alta también tendrán una prioridad alta.

z/OS *Gestor de aplicaciones de crédito (CSQ4CVB2) en z/OS*

El programa Gestor de aplicaciones de crédito (Credit Application Manager, CAM) realiza la mayor parte del procesamiento de la aplicación de comprobación de crédito.

El supervisor desencadenante de CKTI (proporcionado con IBM MQ for z/OS) inicia el CAM cuando se produce un suceso desencadenante en cualquiera de las colas CSQ4SAMP.B2.INQUIRY o cola CSQ4SAMP.B2.REPLY. *n*, donde *n* es un entero que identifica uno de un conjunto de colas de respuestas. El mensaje desencadenante contiene datos que incluyen el nombre de la cola en la que se ha producido el suceso desencadenante.

CAM utiliza colas con nombres en formato CSQ4SAMP.B2.WAITING.n para almacenar información sobre las consultas que está procesando. Las colas están nombradas de forma tal que cada una esté emparejada con una cola de respuesta; por ejemplo, la cola CSQ4SAMP.B2.WAITING.3 contiene los datos de entrada de una determinada consulta y la cola CSQ4SAMP.B2.REPLY.3 contiene un conjunto de mensajes de respuesta (de los programas que consultan bases de datos) relacionados todos con esa misma consulta. Para entender los motivos de este diseño, consulte “Colas de consulta y respuesta independientes en el CAM” en la página 1235.

Lógica de arranque

Si el suceso desencadenante se produce en la cola CSQ4SAMP.B2.INQUIRY, CAM abre la cola para acceso compartido. A continuación, intenta abrir cada cola de respuestas hasta encontrar una cola libre. Si no puede encontrar una cola de respuestas libre, CAM registra el hecho y termina normalmente.

Si el suceso desencadenante se produce en la cola CSQ4SAMP.B2.REPLY.n, CAM abre la cola para acceso exclusivo. Si el código de retorno informa de que el objeto ya está en uso, CAM finaliza con normalidad. Si se produce algún otro error, CAM anota el error y finaliza. CAM abre la cola de espera correspondiente y la cola de consulta y luego empieza a obtener y procesar mensajes. De la cola de espera, CAM recupera detalles de consultas parcialmente completadas.

A efectos de simplicidad, en este ejemplo, los nombres de las colas utilizadas están codificados en el programa. En un entorno empresarial, los nombres de cola probablemente se mantendrían en un archivo al que accedería el programa.

Obtención de un mensaje de la cola de consulta

CAM primero intenta obtener un mensaje de la cola de consulta utilizando la llamada MQGET con la opción MQGMO_SET_SIGNAL. Si un mensaje está disponible inmediatamente, se procesa; si no hay ningún mensaje disponible, se establece una señal.

Luego, CAM intenta obtener un mensaje de la cola de respuestas, usando de nuevo la llamada MQGET con la misma opción. Si un mensaje está disponible de forma inmediata, se procesa; de lo contrario, se establece una señal.

Cuando ambas señales se establecen, el programa espera hasta que se publica una de ellas. Si se publica una señal para indicar que hay un mensaje disponible, este se recupera y procesa. Si la señal caduca o el gestor de colas está terminando, el programa finaliza.

Procesamiento del mensaje recuperado por CAM

Un mensaje recuperado por CAM puede ser de cuatro tipos:

- Un mensaje de consulta.
- Un mensaje de respuesta.
- Un mensaje de propagación.
- Un mensaje inesperado o no deseado.

El CAM procesa estos mensajes, tal como se describe en "Procesamiento del mensaje recuperado por el CAM en z/OS" en la página 1232.

Envío de una respuesta

Cuando CAM ha recibido todas las respuestas que está esperando de una consulta, las procesa y crea un único mensaje de respuesta. Consolida en un mensaje todos los datos de todos los mensajes de respuesta que tienen el mismo `CorrelId`. Esta respuesta se coloca en la cola de respuestas especificada en la solicitud de préstamo original. El mensaje de respuesta se coloca dentro de la misma unidad de trabajo que contiene la recuperación del mensaje de respuesta final. Así se simplifica la recuperación, asegurándose de que nunca haya un mensaje completado en la cola CSQ4SAMP.B2.WAITING.n.

Recuperación de consultas completadas parcialmente

CAM copia en la cola CSQ4SAMP.B2.WAITING.n todos los mensajes que recibe. Establece los campos del descriptor de mensaje así:

- *`Priority`* viene determinado por el tipo de mensaje:
  - En los mensajes de solicitud, prioridad = 3.
  - En datagramas, prioridad = 2.
  - En los mensajes de respuesta, prioridad = 1.
- *`CorrelId`* se establece al *`MsgId`* del mensaje de solicitud de préstamo.
- Otros campos MQMD se copian de los del mensaje recibido.

Cuando se ha completado una consulta, los mensajes de una consulta específica se eliminan de la cola de espera durante el procesamiento de la respuesta. Por lo tanto, en cualquier momento, la cola de espera contiene todos los mensajes relevantes para las consultas en curso. Estos mensajes se utilizan para recuperar los detalles de las consultas en curso si el programa tiene que reiniciarse. Las diferentes prioridades se establecen de modo que los mensajes de consulta se recuperen antes de las propagaciones o mensajes de respuesta.

z/OS *Procesamiento del mensaje recuperado por el CAM en z/OS*

Un mensaje recuperado por el Gestor de aplicaciones de crédito (CAM) puede ser de cuatro tipos. La forma en la que CAM procesa un mensaje depende de su tipo.

Un mensaje recuperado por CAM puede ser de cuatro tipos:

- Un mensaje de consulta.
- Un mensaje de respuesta.
- Un mensaje de propagación.
- Un mensaje inesperado o no deseado.

CAM procesa estos mensajes como se indica a continuación:

Mensaje de consulta

Los mensajes de consulta proceden del programa de interfaz de usuario. Crea un mensaje de consulta por cada solicitud de préstamo.

Por cada solicitud de préstamo, CAM solicita el saldo medio de la cuenta corriente del cliente. Lo hace colocando un mensaje de solicitud en la cola aliasCSQ4SAMP.B2.OUTPUT.ALIAS. Este nombre de cola resuelve a la cola CSQ4SAMP.B3.MESSAGES, que es procesada por el programa de cuentas corrientes CSQ4CVB3. Cuando CAM pone un mensaje en esta cola alias, especifica la correspondiente cola CSQ4SAMP.B2.REPLY.n como cola de respuesta. Aquí se usa una cola alias para que el programa CSQ4CVB3 pueda ser sustituido con facilidad por otro programa que procese una cola base de distinto nombre. Para ello, hay que redefinir la cola alias para que su nombre resuelva a la nueva cola. Además, se puede asignar autorizaciones de acceso diferentes a la cola alias y a la cola base.

Si un usuario solicita un préstamo superior a 10000 unidades, CAM inicia comprobaciones en otras bases de datos también. Lo hace colocando un mensaje de petición en la cola CSQ4SAMP.B4.MESSAGES, que es procesado por el programa de distribución CSQ4CVB4. El proceso que da servicio a esta cola propaga el mensaje a las colas de los programas que tienen acceso a otros registros como, por ejemplo, el historial de tarjetas de crédito, cuentas de ahorro y pagos de hipoteca. Los datos procedentes de estos programas se devuelven a la cola de respuestas especificada en la operación de colocación. Además, este programa envía un mensaje de propagación a la cola de respuestas para especificar cuántos mensajes de propagación se han enviado.

En un entorno de negocio, el programa de distribución probablemente cambiaría el formato de los datos proporcionados para que coincidiera con el formato requerido por cada uno de los otros tipos de cuenta bancaria.

Cualquiera de las colas mencionadas puede estar en un sistema remoto.

Para cada mensaje de consulta, CAM inicia una entrada en la tabla de registros de consulta (IRT) residente en memoria. Este registro contiene:

- El `MsgId` del mensaje de consulta.
- En el campo ReplyExp, el número de respuestas esperadas (igual al número de mensajes enviados).
- En el campo ReplyRec, el número de respuestas recibidas (cero en esta fase).
- En el campo PropsOut, una indicación de si se espera un mensaje de propagación.

CAM copia el mensaje de consulta en la cola de espera con:

- `Priority` establecido a 3.
- `CorrelId` establecido al `MsgId` del mensaje de consulta.
- Los otros campos de descriptor de mensaje establecidos a los correspondientes campos del mensaje de consulta

Mensaje de propagación

Un mensaje de propagación contiene el número de colas a las que el programa de distribución ha reenviado la consulta. El mensaje se procesa del modo siguiente:

1. Suma el número de mensajes enviados al campo ReplyExp del correspondiente registro de la IRT. Esta información está en el mensaje.
2. Se incrementa en 1 el campo ReplyRec del registro de la IRT.
3. Se decrementa en 1 el campo PropsOut del registro de la IRT.
4. Se copia el mensaje en la cola de espera. CAM establece `Priority` a 2 y los demás campos del descriptor de mensaje a los correspondientes campos del mensaje de propagación.

Mensaje de respuesta

Un mensaje de respuesta contiene la respuesta a una de las peticiones al programa de cuentas corrientes o a uno de los programas de consulta de agencia. Los mensajes de respuesta se procesan de la forma siguiente:

1. Se incrementa en 1 el campo ReplyRec del registro de la IRT.
2. Se copia el mensaje en la cola de espera con `Priority` establecido a 1 y con los demás campos del descriptor de mensaje establecidos a los correspondientes campos del mensaje de respuesta.
3. Si ReplyRec = ReplyExp, y PropsOut = 0, se establece el distintivo MsgComplete.

Otros mensajes

La aplicación no espera otros mensajes. No obstante, puede que la aplicación reciba mensajes emitidos por el sistema o mensajes de respuesta con un `CorrelIds` desconocido.

CAM coloca estos mensajes en la cola CSQ4SAMP.DEAD.QUEUE, donde se pueden examinar. Si esta operación de colocación falla, el mensaje se pierde y el programa continúa. Para obtener más información sobre el diseño de esta parte del programa, consulte "Tratamiento de mensajes inesperados en el ejemplo" en la página 1235.

z/OS *Programa de cuenta corriente (CSQ4CVB3) en z/OS*

El programa de cuenta de corriente se inicia con un suceso desencadenante en la cola CSQ4SAMP.B3.MESSAGES. Tras abrir la cola, este programa obtiene un mensaje de ella utilizando la llamada MQGET con la opción de espera y con un intervalo de espera establecido a 30 segundos.

El programa busca en el conjunto de datos VSAM CSQ4BAQ el número de cuenta que aparece en mensaje de solicitud de préstamo. Recupera el nombre de cuenta correspondiente, el saldo medio y el índice de solvencia crediticia, o indica que el número de cuenta no está en el conjunto de datos.

Luego, el programa coloca el mensaje de respuesta (con la llamada MQPUT1) en la cola de respuesta indicada en el mensaje de solicitud de préstamo. Para este mensaje de respuesta, el programa:

- Copia el `CorrelId` del mensaje de solicitud de préstamo.
- Usa la opción MQPMO_PASS_IDENTITY_CONTEXT.

El programa continúa obteniendo mensajes de la cola hasta que vence el intervalo de espera.

z/OS *Programa de distribución (CSQ4CVB4) en z/OS*

El programa de distribución se inicia mediante un suceso desencadenante en la cola CSQ4SAMP.B4.MESSAGES.

Para simular la distribución de la solicitud de préstamo a otras agencias que tienen acceso a registros como, por ejemplo, el historial de tarjetas de crédito, cuentas de ahorro y pagos de hipotecas, el programa coloca una copia del mismo mensaje en todas las colas de la lista de nombres CSQ4SAMP.B4.NAMELIST. Hay tres de estas colas, con nombres con el formato CSQ4SAMP.B *n*.MESSAGES, donde *n* es 5, 6 o 7. En una aplicación empresarial, las agencias podrían estar en ubicaciones separadas, por lo que estas colas podrían ser colas remotas. Si desea modificar la aplicación de ejemplo para mostrarlo, consulte "Ejemplo de comprobación de crédito con varios gestores de colas en z/OS" en la página 1237.

El programa de distribución realiza los pasos siguientes:

1. En la lista de nombres, obtiene los nombres de las colas que el programa va a utilizar. Para ello, el programa utiliza la llamada MQINQ para consultar los atributos del objeto de lista de nombres.

2. Abre las colas y también CSQ4SAMP.B4.MESSAGES.
3. Ejecuta el bucle siguiente hasta que no haya más mensajes en la cola CSQ4SAMP.B4.MESSAGES:
   a. Obtiene un mensaje utilizando la llamada MQGET con la opción de espera y con el intervalo de tiempo de espera establecido en 30 segundos.
   b. Coloca un mensaje en cada cola que aparece en la lista de nombres, especificando el nombre de la cola CSQ4SAMP.B2.REPLY.n correspondiente para la cola de respuesta. El programa copia el *CorrelId* del mensaje de solicitud de préstamo en estos mensajes de copia y utiliza la opción MQPMO_PASS_IDENTITY_CONTEXT en la llamada MQPUT.
   c. Envía un mensaje de datagrama a la cola CSQ4SAMP.B2.REPLY.n para mostrar el número de mensajes que ha podido colocar satisfactoriamente.
   d. Declara un punto de sincronización.

z/OS *Programa de consulta de agencia (CSQ4CVB5/CSQ4CCB5) en z/OS*

El programa de consulta de agencia se proporciona en COBOL y en C. Ambos programas tienen el mismo diseño. Esto muestra que los programas de distintos tipos pueden coexistir dentro de una aplicación IBM MQ y que los módulos de programa que conforman dicha aplicación se pueden sustituir fácilmente.

Una instancia del programa se inicia mediante un suceso desencadenante en cualquiera de estas colas:

- En el programa COBOL (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- En el programa C (CSQ4CCB5), cola CSQ4SAMP.B8.MESSAGES

Nota: Si se desea usar el programa en C, hay que modificar la definición de la lista de nombres CSQ4SAMP.B4.NAMELIST para sustituir la cola CSQ4SAMP.B7.MESSAGES por CSQ4SAMP.B8.MESSAGES. Para ello, puede utilizar cualquiera de los siguientes métodos:

- Los paneles de control y operaciones de IBM MQ for z/OS.
- El comando ALTER NAMELIST.
- La utilidad CSQUTIL.

Tras abrir la correspondiente cola, este programa obtiene un mensaje de ella utilizando la llamada MQGET con la opción de espera y con un intervalo de espera establecido a 30 segundos.

El programa simula la búsqueda de la base de datos de una agencia buscando en el conjunto de datos VSAM CSQ4BAQ el número de cuenta que se ha pasado en el mensaje de solicitud de préstamo. A continuación, crea una respuesta que incluye el nombre de la cola a la que da servicio y un índice de solvencia. Para simplificar el procesamiento, el indice de solvencia se selecciona al azar.

Cuando se coloca el mensaje de respuesta, el programa utiliza la llamada MQPUT1 y:

- Copia el `CorrelId` del mensaje de solicitud de préstamo.
- Usa la opción MQPMO_PASS_IDENTITY_CONTEXT.

El programa envía el mensaje de respuesta a la cola de respuestas indicada en el mensaje de solicitud de préstamo. (El nombre del gestor de colas que es propietario de la cola de respuestas también se especifica en el mensaje de solicitud de préstamo).

z/OS *Consideraciones de diseño para el ejemplo de comprobación de crédito en z/OS*

Consideraciones de diseño en el ejemplo de comprobación de crédito.

Este tema contiene información sobre:

- “Colas de consulta y respuesta independientes en el CAM” en la página 1235
- “Tratamiento de errores en el ejemplo” en la página 1235

- “Tratamiento de mensajes inesperados en el ejemplo” en la página 1235
- “Uso de puntos de sincronización en el ejemplo” en la página 1236
- “Uso de la información de contexto de mensaje en el ejemplo” en la página 1236
- “Uso de identificadores de mensaje y correlación en el CAM” en la página 1237

Colas de consulta y respuesta independientes en el CAM

La aplicación podría utilizar una única cola para consultas y respuestas, pero está diseñada para utilizar colas distintas por los motivos siguientes:

- Cuando el programa está manejando el número máximo de consultas, podrían quedarse consultas adicionales en la cola. Si se utiliza una única cola, esto tendría que sacarse de la cola y almacenarse en otra parte.
- Se podrían iniciar automáticamente otras instancias de CAM para dar servicio a la misma cola de consultas si el tráfico de mensajes es lo suficientemente alto como para justificarlo. Pero el programa tiene que hacer un seguimiento de las consultas en curso y, para ello, tiene que recuperar todas las respuestas a las consultas que ha iniciado. Si solo se utiliza una cola, el programa tendrá que examinar los mensajes para ver si son de este programa o de otro. Esto haría que la operación fuera mucho menos eficiente.

  La aplicación puede soportar varios CAM y puede recuperar las consultas en curso de forma efectiva utilizando colas de espera y de respuesta emparejadas.
- El programa puede esperar en múltiples colas de forma efectiva utilizando señales.

Tratamiento de errores en el ejemplo

El programa de interfaz de usuario maneja los errores notificándoselos directamente al usuario.

Los otros programas no tienen interfaces de usuario, por lo que tienen que manejar los errores de otras maneras. Además, en muchas situaciones (por ejemplo, el fallo de una llamada MQGET), estos programas no conocen la identidad del usuario de la aplicación.

Los otros programas colocan mensajes de error en una cola de almacenamiento temporal CICS llamada CSQ4SAMP. Puede examinar esta cola utilizando la transacción CEBR proporcionada por CICS. Los programas también escriben mensajes de error en el registro CSML de CICS.

Tratamiento de mensajes inesperados en el ejemplo

Cuando se diseña una aplicación de gestión de colas de mensajes, hay que decidir cómo manejar los mensajes que llegan de forma inesperada a una cola.

Las dos opciones básicas son:

- La aplicación no hace más trabajo mientras no se procese el mensaje inesperado. Esto probablemente significa que la aplicación notifica a un operador, termina y garantiza que no se reinicia automáticamente (puede hacerlo desactivando los desencadenantes). Esta opción significa que un único mensaje inesperado puede parar todo el procesamiento de la aplicación y hacer necesaria la intervención de un operador para reiniciarla.
- La aplicación elimina el mensaje de la cola a la que da servicio, coloca el mensaje en otra ubicación y continúa el procesamiento. El mejor lugar para colocar este mensaje es la cola de mensajes no entregados del sistema.

Si selecciona la segunda opción:

- Un operador, u otro programa, debe examinar los mensajes que se colocan en la cola de mensajes no entregados para averiguar de dónde proceden.
- Un mensaje inesperado se pierde si no se puede colocar en la cola de mensajes no entregados.
- Un mensaje inesperado largo se trunca si su longitud supera el límite de los mensajes de la cola de mensajes no entregados o el tamaño de búfer del programa.

Para garantizar que la aplicación maneje sin problemas todas las consultas con un impacto mínimo de actividades externas, la aplicación de ejemplo de comprobación de crédito utiliza la segunda opción. Para permitir que el ejemplo se mantenga separado de otras aplicaciones que utilizan el mismo gestor de colas, el ejemplo de comprobación de crédito no utiliza la cola de mensajes no entregados del sistema; en su lugar, utiliza su propia cola de mensajes no entregados. Esta cola se llama CSQ4SAMP.DEAD.QUEUE. El ejemplo trunca los mensajes cuya longitud supera el área de búfer proporcionada para los programas de ejemplo. Puede utilizar la aplicación de ejemplo Examinar para examinar mensajes en esta cola o utilizar la aplicación de ejemplo Imprimir mensaje para imprimir los mensajes junto con sus descriptores de mensaje.

Sin embargo, si amplía el ejemplo para que ejecute en más de un gestor de colas, el gestor de colas podría colocar en la cola de mensajes no entregados mensajes inesperados o mensajes que no se pueden entregar.

Uso de puntos de sincronización en el ejemplo

Los programas de la aplicación de ejemplo de comprobación de crédito declaran puntos de sincronización para garantizar que:

- Solo se envíe un mensaje de respuesta a cada mensaje esperado.
- Nunca se coloquen múltiples copias de mensajes inesperados en la cola de mensajes no entregados del ejemplo.
- El CAM pueda recuperar el estado de todas las consultas parcialmente completadas obteniendo mensajes permanentes de su cola de espera.

Para lograr esto, se utiliza una única unidad de trabajo para cubrir la obtención de un mensaje, su procesamiento y cualquier operación de colocación posterior.

Uso de la información de contexto de mensaje en el ejemplo

Cuando el programa de interfaz de usuario (CSQ4CVB1) envía mensajes, utiliza la opción MQPMO_DEFAULT_CONTEXT. Esto significa que el gestor de colas genera información de contexto de identidad y de origen. El gestor de colas obtiene esta información de la transacción que ha iniciado el programa (MVB1) y del ID de usuario que ha iniciado la transacción.

Cuando el CAM envía mensajes de consulta, utiliza la opción MQPMO_PASS_IDENTITY_CONTEXT. Esto significa que la información de contexto de identidad del mensaje que se está colocando se copia del contexto de identidad del mensaje de consulta original. Con esta opción, la información de contexto de origen la genera el gestor de colas.

Cuando el CAM envía mensajes de respuesta, utiliza la opción MQPMO_ALTERNATE_USER_AUTHORITY. Esto provoca que el gestor de colas use un ID de usuario alternativo en su comprobación de seguridad cuando el CAM abre una cola de respuesta. El CAM utiliza el ID de usuario del emisor del mensaje de consulta original. Esto significa que a los usuarios solo se les permite ver las respuestas a las consultas que ellos han originado. El ID de usuario alternativo se obtiene a partir de la información de contexto de identidad del descriptor del mensaje de consulta original.

Cuando los programas de consulta (CSQ4CVB3/4/5) envían mensajes de respuesta, utilizan la opción MQPMO_PASS_IDENTITY_CONTEXT. Esto significa que la información de contexto de identidad del mensaje que se está colocando se copia del contexto de identidad del mensaje de consulta original. Con esta opción, la información de contexto de origen la genera el gestor de colas.

Nota: El ID de usuario asociado a las transacciones MVB3/4/5 requiere acceso a las colas B2.REPLY.n. Puede que estos ID de usuario no sean los mismos que los asociados a la solicitud que se está procesando. Para soslayar esta posible vulnerabilidad de seguridad, los programas de consulta podrían utilizar la opción MQPMO_ALTERNATE_USER_AUTHORITY al colocar sus respuestas. Esto significaría que cada usuario individual de MVB1 necesitaría autorización para abrir las colas B2.REPLY.n.

Uso de identificadores de mensaje y correlación en el CAM

La aplicación tiene que supervisar el progreso de todas las consultas activas que está procesando en un momento dado. Para ello, utiliza el identificador de mensaje exclusivo de cada mensaje de solicitud de préstamo para asociar toda la información que tiene de cada consulta.

El CAM copia el mensaje `MsgId` del mensaje de consulta en el `CorrelId` de todos los mensajes de solicitud que envía para dicha consulta. Los otros programas del ejemplo (CSQ4CVB3 - 5) copian el `CorrelId` de cada mensaje que reciben en el `CorrelId` de su mensaje de respuesta.

z/OS *Ejemplo de comprobación de crédito con varios gestores de colas en z/OS*

Se puede usar la aplicación de comprobación de crédito de ejemplo para ilustrar un encolamiento distribuido instalando dicho ejemplo en dos gestores de colas y sistemas CICS (con cada gestor de colas conectado con un sistema CICS distinto).

Cuando se instala el programa de ejemplo y el supervisor desencadenante (CKTI) ejecuta en cada sistema, haga lo siguiente:

1. Configure el enlace de comunicación entre ambos gestores de colas. Para obtener información sobre cómo hacer esto, consulte Configuración de encolamientos distribuidos.
2. En un gestor de colas, cree una definición local por cada una de las colas remotas (en el otro gestor de colas) que desee utilizar. Estas colas pueden ser cualquiera de CSQ4SAMP.B *n*.MESSAGES, donde *n* es 3, 5, 6 o 7. (Estas son las colas a las que da servicio el programa de cuentas corrientes y el programa de consulta de agencia). Para obtener información sobre cómo hacerlo, consulte DEFINE QREMOTE y Colas DEFINE.
3. Cambie la definición de la lista de nombres (CSQ4SAMP.B4.NAMELIST) para que contenga los nombres de las colas remotas que desee utilizar. Para obtener información sobre cómo realizar esta operación, consulte DEFINE NAMELIST.

z/OS *Extensión IMS del ejemplo de comprobación de crédito en z/OS*

Se proporciona una versión del programa de comprobación de cuentas como un programa de procesamiento de mensajes por lotes (BMP) de IMS. Está escrito en C.

El programa realiza la misma función que la versión de CICS, excepto que para obtener la información de cuenta, el programa lee una base de datos IMS en un lugar de un archivo VSAM. Si sustituye la versión de CICS del programa de comprobación de cuenta con la versión de IMS, no verá ninguna diferencia en el método de utilización de la aplicación.

Para preparar y ejecutar la versión de IMS, debe:

1. Siga los pasos de “Preparación y ejecución del programa de ejemplo Credit Check (Comprobación de crédito) en z/OS” en la página 1226.
2. Siga los pasos de “Preparación de la aplicación de ejemplo para el entorno IMS en z/OS” en la página 1206.
3. Modificar la definición de la cola alias CSQ4SAMP.B2.OUTPUT.ALIAS para que resuelva a la cola CSQ4SAMP.B3.IMS.MESSAGES (en lugar de CSQ4SAMP.B3.MESSAGES). Para ello, puede utilizar uno de los siguientes:
   - Los paneles de control y operaciones de IBM MQ for z/OS.
   - El comando ALTER QALIAS.

Otra forma de usar el programa de comprobación de cuentas corrientes de IMS es hacer que de servicio a una de las colas que recibe mensajes del programa de distribución. En la entrega de la aplicación de ejemplo de comprobación de crédito, hay tres de estas colas (B5/6/7.MESSAGES), todas ellas servidas por el programa de consulta de agencia. Este programa busca en un conjunto de datos VSAM. Para comparar el uso del conjunto de datos VSAM y la base de datos IMS, podría hacer que el programa de comprobación de cuenta de IMS preste servicio a una de estas colas en su lugar. Para ello, debe modificar la definición de la lista de nombres CSQ4SAMP.B4.NAMELIST para sustituir una de las colas CSQ4SAMP.B *n*.MESSAGES con la cola CSQ4SAMP.B3.IMS.MESSAGES. Puede utilizar uno de los siguientes:

- Los paneles de control y operaciones de IBM MQ for z/OS.
- El comando ALTER NAMELIST.

A continuación, puede ejecutar el ejemplo desde la transacción de CICS MVB1. El usuario no apreciará ninguna diferencia en la operación ni en la respuesta. El BMP IMS se detiene después de recibir un mensaje de parada o después de estar inactivo durante cinco minutos.

Diseño del programa de comprobación de cuenta de IMS (CSQ4ICB3)

Este programa ejecuta como un BMP. Inicie el programa utilizando su JCL antes de que se le envía ningún mensaje IBM MQ.

El programa busca en una base de datos IMS el número de cuenta de los mensajes de solicitud de préstamo. Recupera el nombre de cuenta correspondiente, el saldo medio y el índice de solvencia crediticia.

El programa envía los resultados de la búsqueda de la base de datos a la cola de respuesta indicada en el mensaje IBM MQ que se está procesando. El mensaje devuelto añade el tipo de cuenta y el resultado de la búsqueda al mensaje recibido para que la transacción que crea la respuesta pueda confirmar que se está procesando la consulta correcta. El mensaje tiene un formato de tres grupos de 79 caracteres, como se indica a continuación:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Cuando se ejecuta un BMP orientado a mensajes, el programa drena la cola de mensajes de IMS desde la cola IBM MQ for z/OS y los procesa. No se recibe información de la cola de mensajes IMS. El programa se vuelve a conectar con el gestor de colas tras cada punto de comprobación, porque se han cerrado los manejadores.

Cuando ejecuta en un BMP orientado a lotes, el programa sigue conectado con el gestor de colas tras cada punto de comprobación, porque los descriptores no se cierran.

z/OS *Ejemplo del manejador de mensajes en z/OS*

La aplicación TSO de ejemplo de manejador de mensajes permite examinar, reenviar y borrar mensajes en una cola. El ejemplo está disponible en C y COBOL.

Preparación y ejecución del ejemplo

Siga estos pasos:

1. Prepare el ejemplo tal como se describe en “Preparación de aplicaciones de ejemplo para el entorno TSO en z/OS” en la página 1200.
2. Adapte la CLIST (CSQ4RCH1) proporcionada para el ejemplo para definir la ubicación de los paneles, la ubicación del archivo de mensaje y la ubicación de los módulos de carga.

Puede utilizar CLIST CSQ4RCH1 para ejecutar tanto la versión en C como la versión en COBOL del ejemplo. La versión proporcionada de CSQ4RCH1 ejecuta la versión en C y contiene instrucciones para la adaptación necesaria en la versión en COBOL.

Nota:

1. En el ejemplo no se proporciona ninguna definición de cola de ejemplo.
2. VS COBOL II no soporta multitarea con ISPF; por tanto, no utilice la aplicación de ejemplo de manejador de mensajes en ambos lados de una pantalla dividida. Si lo hace, los resultados serán impredecibles.

z/OS *Utilización del ejemplo del manejador de mensajes en z/OS*

Una vez instalado e invocado el ejemplo desde el CLIST CSQ4RCH1 adaptado, aparecerá la pantalla mostrada en Figura 146 en la página 1239.

```
---------------------- IBM MQ for z/OS -- Samples ----------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :


Queue Manager Name  : ________________________________________________ :

Queue Name      : _____________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figura 146. Pantalla inicial del ejemplo de manejo de mensajes

Especifique el gestor de colas y el nombre de cola que se va a visualizar (se distingue entre mayúsculas y minúsculas) y se muestra la pantalla de la lista de mensajes (consulte Figura 147 en la página 1239).

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figura 147. Pantalla de lista de mensajes del ejemplo de manejo de mensajes

Esta pantalla muestra los primeros 99 mensajes en la cola y, por cada uno de ellos, muestra los campos siguientes:

Msg No
Número de mensaje

Put Date MM/DD/YYYY
Fecha en que el mensaje se ha colocado en la cola (GMT)

Put Time HH:MM:SS
Hora en que el mensaje se ha colocado en la cola (GMT)

Format Name
Campo MQMD.Format

User Identifier
Campo MQMD.UserIdentifier

Put Application Type
Campo MQMD.PutApplType

Put Application Name
Campo MQMD.PutApplName

También se visualiza el número total de mensajes en la cola.

En esta pantalla se puede elegir un mensaje por número (no por posición de cursor) y visualizarse. Para obtener un ejemplo, consulte Figura 148 en la página 1240.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                       :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :
Forward to Q Mgr : VM03                        :
Forward to Queue : QL.TEST.ISCRES1                  :

Action : _ :  (D)elete (F)orward

Message Content :
------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount   : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr   : `VM03                        `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                    `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .....T....`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figura 148. Se visualiza el mensaje seleccionado

Una vez que se ha visualizado el mensaje, se puede borrar, dejar en la cola o reenviarse a otra cola. Los campos `Forward to Q Mgr` (reenviar a gestor de colas y `Forward to Queue` (reenviar a cola) se inicializan a partir del MQMD y se pueden cambiar antes de reenviar el mensaje.

El diseño de ejemplo sólo permite que se seleccionen y visualicen mensajes con combinaciones `MsgId` / `CorrelId` exclusivas, porque el mensaje se recupera utilizando `MsgId` y `CorrelId` como clave. Si la clave no es exclusiva, el ejemplo no podrá recuperar con certeza el mensaje seleccionado.

Nota: Cuando se usa el ejemplo SCSQCLST(CSQ4RCH1) para examinar mensajes, cada invocación hace que aumente el recuento de restituciones del mensaje. Si desea cambiar el comportamiento de este ejemplo, haga una copia y modifique el contenido según sea necesario. Debe tener en cuenta que

otras aplicaciones que se basen en este recuento de restituciones puedan verse influenciadas por este incremento del recuento.

z/OS *Diseño del ejemplo de manejo de mensajes de ejemplo en z/OS*

En este tema se describe el diseño de cada uno de los programas de que consta la aplicación de ejemplo de manejo de mensajes.

Programa de validación de objetos

Solicita una cola y un nombre de gestor de colas válidos.

Si no se especifica un nombre de gestor de colas, se utilizará el gestor de colas predeterminado si está disponible. Solo se pueden usar colas locales; se emite un MQINQ para comprobar el tipo de cola y se genera un error si esta no es local. Si la cola no se abre satisfactoriamente, o la llamada MQGET está inhibida en la cola, se devuelven mensajes de error que indican los códigos de retorno de razón y de CompCode.

Programa de lista de mensajes

Muestra una lista de los mensajes de una cola con información sobre ellos como, por ejemplo, putdate (fecha de colocación), puttime (hora de colocación) y el formato del mensaje.

El número máximo de mensajes almacenados en la lista es de 99. Si hubiera más mensajes en la cola, también se mostraría la profundidad de cola actual. Para elegir el mensaje que se va a visualizar, escriba su número en el campo de entrada (el valor predeterminado es 01). Si la entrada no es válida, recibirá el correspondiente mensaje de error.

Programa de contenido de mensajes

Muestra el contenido de mensajes.

El contenido se formatea y se divide en dos partes:

1. Descriptor de mensaje
2. Búfer del mensaje.

El descriptor de mensaje muestra el contenido de cada campo en una línea aparte.

El búfer del mensajes se formatea en función de su contenido. Si el búfer contiene una cabecera de mensaje no entregado (MQDLH) o una cabecera de cola de transmisión (MQXQH), estas se formatean y visualizan antes del propio búfer.

Antes de que los datos del se formateen, una línea de título muestra la longitud del búfer del mensaje en bytes. El tamaño máximo del búfer es de 32768 bytes y cualquier mensaje de mayor longitud se trunca. Se visualiza el tamaño completo del búfer junto con un mensaje que indica que solo se visualizan los primeros 32768 bytes del mensaje.

Los datos del búfer se formatean de dos maneras:

1. Una vez impreso el desplazamiento en el búfer, sus datos se visualizan en hexadecimal.
2. A continuación, los datos del búfer se visualizan de nuevo como valores EBCDIC. Si algún valor EBCDIC no se puede imprimir, se imprime un punto (.) en su lugar.

En el campo de acción, puede especificar D para borrar o F para reenviar. Si elige reenviar el mensaje, hay que configurar correctamente la `forward-to queue` (cola de reenvío) y el `queue manager name` (nombre del gestor de colas). Los valores predeterminados de estos campos se leen de los campos ReplyToQ y ReplyToQMgr del descriptor de mensaje.

Si reenvía un mensaje, se elimina cualquier bloque de cabecera almacenado en el búfer. Si el mensaje se reenvía satisfactoriamente, se elimina de la cola original. Si se especifican acciones no válidas, aparecen mensajes de error.

También está disponible un ejemplo de panel de ayuda llamado CSQ4CHP9.

z/OS *Ejemplo de colocación asíncrona en z/OS*

El programa de ejemplo de colocación asíncrona coloca mensajes en una cola utilizando la llamada MQPUT asíncrona. Este ejemplo también recupera información de estado con la llamada MQSTAT.

Una aplicación de colocación asíncrona usa las siguientes llamadas MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

Los programas de ejemplo se entregan en C.

Las aplicaciones de colocación asíncrona se ejecutan en el entorno de procesamiento por lotes. Consulte la sección Otros ejemplos para las aplicaciones por lotes.

En este tema también se proporciona información sobre el diseño del programa de consumo asíncrono y la ejecución del ejemplo CSQ4BCS2.

- "Ejecución del ejemplo CSQ4BCS2" en la página 1242
- "Diseño del programa de ejemplo de colocación asíncrona." en la página 1242

Ejecución del ejemplo CSQ4BCS2

Este programa de ejemplo recibe hasta seis parámetros:

1. El nombre de la cola de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Opciones de apertura (opcional).
4. Opciones de cierre (opcional).
5. El nombre del gestor de colas de destino (opcional).
6. El nombre de la cola dinámica (optional).

Si no se especifica ningún gestor de colas, CSQ4BCS2 se conecta con el gestor de colas predeterminado. El contenido del mensaje se facilita a través de la entrada estándar (**SYSIN DD**).

Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BCSP.

Diseño del programa de ejemplo de colocación asíncrona.

El programa usa la llamada MQOPEN con cualquiera de las opciones de salida proporcionadas, o con las opciones MQOO_OUTPUT y MQOO_FAIL_IF_QUIESCING, para abrir la cola de destino y colocar en ella mensajes.

Si el programa no puede abrir la cola, genera un mensaje de error que contiene el código de razón devuelto por la llamada MQOPEN. Para mantener el programa simple en esta llamada MQI y sucesivas, se utilizan valores predeterminados en muchas de las opciones.

Para cada línea de entrada, el programa lee el texto en un búfer y utiliza la llamada MQPUT con MQPMO_ASYNC_RESPONSE para crear un mensaje de datagrama que contiene el texto de dicha línea, y coloca el mensaje asíncronamente en la cola de destino. El programa continúa hasta llegar final de la entrada, o hasta que falla la llamada MQPUT. Si el programa alcanza el final de la entrada, cierra la cola con la llamada MQCLOSE.

A continuación, el programa emite la llamada MQSTAT, que devuelve una estructura MQSTS, y muestra mensajes que contienen el número de mensajes colocados correctamente, el número de mensajes colocados con un aviso y el número de errores.

Nota: Para observar lo que sucede cuando la llamada MQSTAT detecta un error de MQPUT, establezca MAXDEPTH en la cola de destino a un valor bajo.

z/OS *El ejemplo de consumo asíncrono por lotes en z/OS*

El programa de ejemplo CSQ4BCS1 se entrega en C y muestra el uso de MQCB y MQCTL para consumir mensajes desde varias colas de forma asíncrona.

Los ejemplos de consumo asíncrono se ejecutan en el entorno de proceso por lotes. Consulte la sección Otros ejemplos para las aplicaciones por lotes.

También existe un ejemplo COBOL que se ejecuta en el entorno CICS, consulte la sección "El ejemplo de consumo asíncrono y de publicación/suscripción de CICS en z/OS" en la página 1244.

Las aplicaciones utilizan estas llamadas MQI:

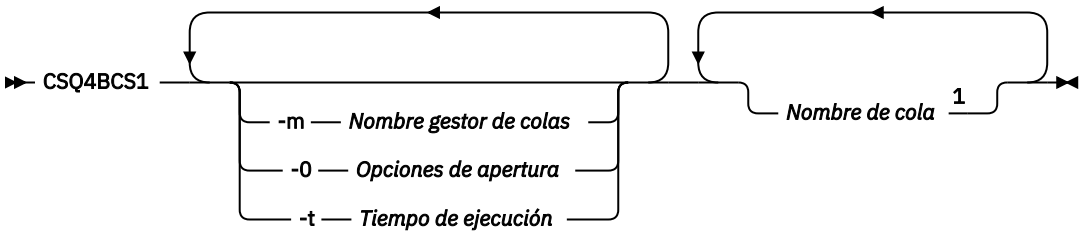
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

Este tema también proporciona información acerca de las cabeceras siguientes:

- "Ejecución del ejemplo CSQ4BCS1" en la página 1243
- "Diseño del programa de ejemplo de consumo asíncrono por lotes" en la página 1243

Ejecución del ejemplo CSQ4BCS1

Este programa sigue la sintaxis siguiente:



Notas:

[1] Este ejemplo da soporte a un máximo de 10 colas. Proporcione más de un nombre de cola para leer los mensajes de varias colas.

Existe un JCL de ejemplo para ejecutar este programa que se encuentra en CSQ4BCSC.

Diseño del programa de ejemplo de consumo asíncrono por lotes

El ejemplo muestra cómo leer mensajes de varias colas en el orden de su llegada. Para ello es necesario más código utilizando MQGET síncrono. Con el consumo asíncrono, no es necesario el sondeo y la gestión de las hebras y del almacenamiento la lleva a cabo IBM MQ. En el programa de ejemplo, los errores se muestran en la consola.

El código de ejemplo tiene los pasos siguientes:

1. Definir la función de devolución de llamada de consumo de mensaje único.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Conectar con el gestor de colas.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Abrir las colas de entrada y asociar cada cola a la función de devolución de llamada `MessageConsumer`.

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   No es necesario configurar `cbd.CallbackFunction` en cada cola; es un campo de solo entrada. Puede asociar una función de devolución de llamada diferente a cada cola.

4. Iniciar el consumo de los mensajes.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Esperar a que el usuario pulse Intro y, a continuación, detener el consumo de mensajes.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Finalmente, desconectar del gestor de colas.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *El ejemplo de consumo asíncrono y de publicación/suscripción de CICS en z/OS*

Programas de ejemplo de consumo asíncrono y de publicación/suscripción, y las características de publicación y suscripción dentro de CICS.

Un programa *cliente de registro* registra tres manejadores de devolución de llamada (un manejador de sucesos y dos consumidores de mensajes), e inicia el consumo asíncrono. Un programa *cliente de mensajería* pone mensajes en la cola, o publica mensajes adecuados desde una consola CICS para su consumo por parte de los dos consumidores de mensajes (CSQ4CVCN y CSQ4CVCT).

Para proporcionar control en tiempo de ejecución sobre el comportamiento del ejemplo, se puede dar instrucciones a uno de los consumidores de mensajes utilizando los mensajes que recibe, para suspender (SUSPEND), reanudar (RESUME) o anular el registro (DEREGISTER) de cualquier de los manejadores de devolución de llamada. También se puede utilizar para emitir un MQCTL STOP para poner fin al consumo asíncrono bajo control. El otro consumidor de mensajes se registra para suscribirse a un tema.

Cada programa emite sentencias COBOL DISPLAY en los puntos adecuados para mostrar el comportamiento del ejemplo.

Las aplicaciones utilizan estas llamadas MQI:

- MQOPEN
- MQPUT
- MQSUB

- MQGET
- MQCLOSE
- MQCB
- MQCTL

Los programas se entregan en el lenguaje COBOL. Consulte Ejemplos de consumo asíncrono y publicación/suscripción de CICS para las aplicaciones CICS.

En este tema se proporciona también la información siguiente:

- "Configuración" en la página 1245
- "Cliente de registro CSQ4CVRG" en la página 1245
- "Manejador de sucesos CSQ4CVEV" en la página 1245
- "Consumidor de mensajes simple CSQ4CVCN" en la página 1246
- "Consumidor de mensajes de control CSQ4CVCT" en la página 1246
- "Cliente de mensajería CSQ4CVPT" en la página 1246

Configuración

Los nombres de la cola y el tema utilizados por los consumidores de mensajes están codificados en los programas cliente de registro y mensajería.

La cola, **SAMPLE.CONTROL.QUEUE**, debe estar definida en el gestor de colas asociado con la región CICS antes de ejecutar el ejemplo. El tema, **Noticias/Medios/Películas**, se puede definir si es necesario, o se puede crear en tiempo de ejecución bajo el objeto administrativo predeterminado si no existe.

Los programas y definiciones de transacción de CICS se pueden instalar mediante la instalación de un grupo: CSQ4SAMP.

Cliente de registro CSQ4CVRG

El programa de cliente de registro se debe iniciar bajo la transacción CICS MVRG. No acepta ninguna entrada.

Cuando se inicia, el cliente de registro registra los manejadores de devolución de llamada siguientes utilizando MQCB:

- CSQ4CVEV como un manejador de sucesos.
- CSQ4CVCN como un consumidor de mensajes en un tema, **Noticias/Medios/Películas**.
- CSQ4CVCT como un consumidor de mensajes en una cola, **SAMPLE.CONTROL.QUEUE**.

El cliente de registro pasa una estructura de datos que contiene los nombres de los tres manejadores de devolución de llamada registrados a CSQ4CVCT, junto con los manejadores de objetos asociados a los dos consumidores de mensajes.

Habiendo registrado los manejadores de devolución de llamada, el cliente de registro emite MQCTL START_WAIT para iniciar el consumo asíncrono, y se suspende hasta que se le devuelva el control (por ejemplo, cuando uno de los manejadores de devolución de llamada emita MQCTL STOP).

Manejador de sucesos CSQ4CVEV

Cuando está controlado, el manejador de sucesos muestra un mensaje que indica el tipo de llamada (por ejemplo, START). Cuando se controla para el código de razón CONNECTION_QUIESCING de IBM MQ, el manejador de sucesos emite MQCTL STOP para finalizar el consumo asíncrono y devolver el control al cliente de registro.

Consumidor de mensajes simple CSQ4CVCN

Cuando está controlado, este consumidor de mensajes muestra un mensaje que indica el tipo de llamada (por ejemplo, REGISTER). Cuando se controla para el tipo de llamada MSG_REMOVED, el consumidor de mensajes recupera el mensaje de entrada y lo envía al registro de trabajo de CICS .

Consumidor de mensajes de control CSQ4CVCT

Cuando está controlado, este consumidor de mensajes muestra un mensaje que indica el tipo de llamada (por ejemplo, START). Cuando se controla para el tipo de llamada MSG_REMOVED, el consumidor de mensajes recupera el mensaje de entrada y la estructura de datos que pasa el cliente de registro. En función del contenido del mensaje, emite el mandato MQCB o MQCTL adecuado a uno de los siguientes:

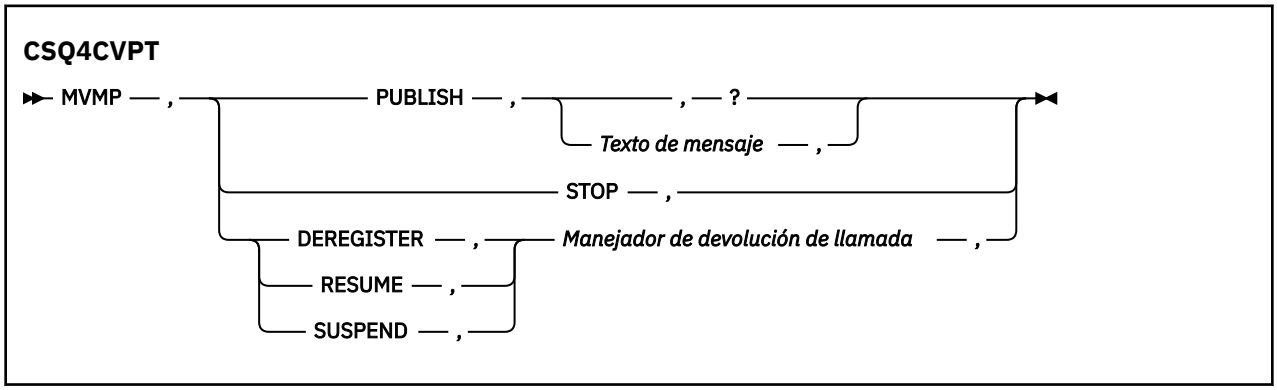
- Detener (STOP) el consumo asíncrono (devolviendo el control al cliente de registro).
- Suspender (SUSPEND), reanudar (RESUME) o anular el registro (DEREGISTER) de un manejador de devolución de llamada denominado (incluyéndose a sí mismo).

Cliente de mensajería CSQ4CVPT

El cliente de mensajería tiene dos funciones:

- Publica un mensaje en un tema para su consumo por parte del consumidor de mensajes CSQ4CVCN.
- Pone un mensaje de control en una cola para su consumo por parte del consumidor de mensajes de control CSQ4CVCT, provocando un cambio potencial en el comportamiento del ejemplo.

El programa del cliente de mensajería debe iniciarse desde una consola CICS bajo una transacción CICS, y recibe una entrada de línea de mandatos con la sintaxis siguiente:



PUBLISH
Publicar el texto del mensaje (o un mensaje predeterminado) como un mensaje retenido para su consumo por parte del consumidor de mensajes simple.

STOP
Detener el consumo asíncrono.

DEREGISTER
Anular el registro del manejador de devolución de llamada denominado.

RESUME
Reanudar del manejador de devolución de llamada denominado.

SUSPEND
Suspender el manejador de devolución de llamada denominado.

Los campos de entrada son posicionales y están separados por comas. Las palabras clave y los nombres de manejador de devolución de llamada no distinguen entre mayúsculas y minúsculas.

Ejemplos:

Tabla 186. Ejemplos de entrada	
Ejemplo	**Descripción**
MVMP,PUBLISH,,	Publicar un mensaje predeterminado
MVMP,publish, `A short message`,	Publicar el texto proporcionado
MVMP,STOP,	Detener el consumo asíncrono
MVMP,DEREGISTER,CSQ4CVEV,	Anular el registro del manejador de sucesos
MVMP,resume,csq4cvcn,	Reanudar el consumidor de mensajes simple
MVMP,SUSPEND,CSQ4CVEV,	Suspender el manejador de sucesos

Donde MVMP es la transacción CICS asociada con el programa cliente de mensajería CSQ4CVPT.

Nota:

- La suspensión o la anulación del registro de todos los manejadores de devolución de llamada termina la llamada START_WAIT emitida por el cliente de registro, devolviendo el control a este y finalizando la tarea.
- La suspensión o la anulación del registro del manejador de devolución de llamada de control no se ha evitado deliberadamente, pero elimina la posibilidad de controlar más el comportamiento del ejemplo.

z/OS *Ejemplo de publicación/suscripción en z/OS*

Los programas de ejemplo de publicación/suscripción muestran el uso de las características de publicación y suscripción en IBM MQ.

Hay cuatro programas de ejemplo de lenguaje de programación C y dos de lenguaje de programación COBOL que demuestran cómo programar la interfaz de publicación/suscripción de IBM MQ. Los programas se entregan en C y COBOL. Las aplicaciones se ejecutan en el entorno de procesamiento por lotes; consulte Ejemplos de publicación/suscripción para obtener las aplicaciones por lotes.

También hay ejemplos de COBOL que se ejecutan en el entorno CICS; consulte “El ejemplo de consumo asíncrono y de publicación/suscripción de CICS en z/OS” en la página 1244.

Este tema también proporciona información sobre cómo ejecutar programas de ejemplo de publicación/suscripción. Estos programas de ejemplo incluyen:

- “Ejecución del ejemplo CSQ4BCP1” en la página 1247
- “Ejecución del ejemplo CSQ4BCP2” en la página 1248
- “Ejecución del ejemplo CSQ4BCP3” en la página 1248
- “Ejecución del ejemplo CSQ4BCP4” en la página 1248
- “Ejecución del ejemplo CSQ4BVP1” en la página 1249
- “Ejecución del ejemplo CSQ4BVP2” en la página 1249

Ejecución del ejemplo CSQ4BCP1

Este programa está escrito en C; publica mensajes en un tema. Inicie uno de los ejemplos de suscriptor antes de ejecutar este programa.

Este programa recibe hasta cuatro parámetros:

1. El nombre de la cadena de tema de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Opciones de apertura (opcional).
4. Opciones de cierre (opcional).

Si no se especifica ningún gestor de colas, CSQ4BCP1 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BCPP.

El contenido del mensaje se facilita a través de la entrada estándar (**SYSIN DD**).

Ejecución del ejemplo CSQ4BCP2

Este programa está escrito en C; se suscribe a un tema e imprime el mensaje recibido.

Este programa recibe hasta tres parámetros:

1. El nombre de la cadena de tema de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Opciones de suscripción MQSD (opcional).

Si no se especifica ningún gestor de colas, CSQ4BCP2 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BCPS.

Ejecución del ejemplo CSQ4BCP3

Este programa está escrito en C; se suscribe a un tema usando un destino especificado por el usuario e imprime el mensaje recibido.

Este programa recibe hasta cuatro parámetros:

1. El nombre de la cadena de tema de destino (obligatorio).
2. El nombre del destino (obligatorio).
3. El nombre del gestor de colas (opcional).
4. Opciones de suscripción MQSD (opcional).

Si no se especifica ningún gestor de colas, CSQ4BCP3 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BCPD.

Ejecución del ejemplo CSQ4BCP4

Este programa está escrito en C; se suscribe y obtiene mensajes de un tema que permite el uso de opciones ampliadas en la llamada MQSUB, ampliando las que están disponibles en el ejemplo MQSUB más simple: CSQ4BCP2. Además de la carga útil del mensaje, se reciben y visualizan las propiedades de cada mensaje.

Este programa recibe un conjunto variable de parámetros:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  Importante: Se necesita uno de **-t** o **-o**, o ambos,
- **-m** *Queue manager name* (Opcional).
- **-b** *Connection binding type* (opcional), donde *type* puede tener cualquiera de los valores siguientes:
  - *standard* : MQCNO_STANDARD_BINDING, que es el valor predeterminado
  - *shared*: MQCNO_SHARED_BINDING
  - *vía de acceso rápida*: MQCNO_FASTPATH_BINDING
  - *aislado*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (Opcional).
- **-w** *Wait interval on MQGET in seconds* (opcional), donde *segundos* puede tener cualquiera de los valores siguientes:
  - unlimited: MQWI_UNLIMITED

  - none: Sin espera
  - *n*: Intervalo de espera en segundos
  - Ningún valor especificado: Cuando no se especifica ningún valor, el valor predeterminado es de 30 segundos
- **-d** *Subscription name* (Opcional). Crea o reanuda una suscripción duradera nombrada.
- **-k** (Opcional). Mantiene la suscripción duradera al hacer un MQCLOSE.

Si no se especifica ningún gestor de colas, CSQ4BCP4 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BCPE.

Ejecución del ejemplo CSQ4BVP1

Este programa está escrito en COBOL, publica mensajes en un tema. Inicie uno de los ejemplos de suscriptor antes de ejecutar este programa.

Este programa no recibe ningún parámetro. **SYSIN DD** proporciona el nombre de tema de entrada, el nombre del gestor de colas y el contenido del mensaje.

Si no se especifica ningún gestor de colas, CSQ4BVP1 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BVPP.

Ejecución del ejemplo CSQ4BVP2

Este programa está escrito en COBOL, se suscribe a un tema e imprime el mensaje recibido.

Este programa no recibe ningún parámetro. **SYSIN DD** proporciona la entrada para el nombre de tema y el nombre del gestor de colas.

Si no se especifica ningún gestor de colas, CSQ4BVP1 se conecta con el gestor de colas predeterminado. Hay un JCL de ejemplo para ejecutar el programa y que reside en CSQ4BVPP.

z/OS *Ejemplo de establecimiento y consulta de las propiedades de un mensaje en z/OS*

Los programas de ejemplo de propiedades de mensaje ilustran la adición de propiedades definidas por el usuario a un descriptor de mensaje y la consulta de las propiedades asociadas a dicho mensaje.

Las aplicaciones utilizan estas llamadas MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

Los programas se entregan en C. Las aplicaciones ejecutan en el entorno de procesamiento por lotes. Consulte la sección Otros ejemplos para las aplicaciones por lotes.

El programa CSQ4BCM1 se utiliza para consultar las propiedades de un descriptor de mensaje de una cola de mensajes, y es un ejemplo del uso de la llamada de API MQINQMP. El ejemplo obtiene un mensaje de una cola y e imprime todas las propiedades del descriptor de mensaje.

El programa CSQ4BCM2 se utiliza para establecer las propiedades de un descriptor de mensaje en una cola de mensajes, y es un ejemplo del uso de la llamada de API MQSETMP. El ejemplo crea un descriptor

de mensaje y lo coloca en el campo `MsgHandle` de la estructura MQGMO. Luego coloca el mensaje en una cola.

En los programas CSQ4BCG1 y CSQ4BCP4 se incluyen otros ejemplos de consulta e impresión de propiedades de mensaje.

En este tema también se proporciona información sobre la ejecución de los ejemplos de establecimiento y consulta de las propiedades de un mensaje bajo las cabeceras siguientes:

- "Ejecución del ejemplo CSQ4BCM1" en la página 1250
- "Ejecución del ejemplo CSQ4BCM2" en la página 1250

Ejecución del ejemplo CSQ4BCM1

Este programa recibe hasta cuatro parámetros:

1. El nombre de la cola de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Opciones de apertura (opcional).
4. Opciones de cierre (opcional).

Ejecución del ejemplo CSQ4BCM2

Este programa recibe hasta seis parámetros:

1. El nombre de la cola de destino (obligatorio).
2. El nombre del gestor de colas (opcional).
3. Opciones de apertura (opcional).
4. Opciones de cierre (opcional).
5. El nombre del gestor de colas de destino (opcional).
6. El nombre de la cola dinámica (optional).

Los nombres de propiedad, los valores y el contenido del mensaje se proporcionan a través de la entrada estándar (**`SYSIN DD`**). Hay un JCL de ejemplo para ejecutar el programa, que reside en CSQ4BCMP.

Desarrollo de aplicaciones para Managed File Transfer

Especifique los programas que se van a ejecutar con Managed File Transfer, utilice Apache Ant con Managed File Transfer, personalice Managed File Transfer con salidas de usuario y controle Managed File Transfer colocando los mensajes en la cola de mandatos de agente.

Especificación de programas que se van a ejecutarse con MFT

Se pueden ejecutar programas en un sistema en el que se esté ejecutando un Managed File Transfer Agent. Como parte de una solicitud de transferencia de archivos, puede especificar un programa para que se ejecute antes de que se inicie una transferencia, o después de que ésta finalice. Además, puede iniciar un programa que no forme parte de una solicitud de transferencia de archivos sometiendo una solicitud de llamada gestionada.

Acerca de esta tarea

Hay cinco escenarios en los que puede especificar un programa para que se ejecute:

- Como parte de una solicitud de transferencia, en el agente de origen, antes de que se inicie la transferencia.
- Como parte de una solicitud de transferencia, en el agente de destino, antes de que se inicie la transferencia.
- Como parte de una solicitud de transferencia, en el agente de origen, una vez finalizada la transferencia.

- Como parte de una solicitud de transferencia, en el agente de destino, una vez finalizada la transferencia.
- No como parte de una solicitud de transferencia. Puede someter una solicitud a un agente para que ejecute un programa. Este escenario a veces se denomina llamada gestionada.

Las salidas de usuario y las llamadas de programa se invocan en el orden siguiente:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Notas:

1. El **DestinationTransferEndExits** sólo se ejecuta cuando finaliza la transferencia, ya sea de forma satisfactoria o parcial.
2. El **postDestinationCall** sólo se ejecuta cuando finaliza la transferencia, ya sea de forma satisfactoria o parcial.
3. El **SourceTransferEndExits** se ejecuta para transferencias satisfactorias, parcialmente satisfactorias o anómalas.
4. **postSourceCall** sólo se llama si:
   - La transferencia no se ha cancelado.
   - Hay un resultado satisfactorio o parcialmente satisfactorio.
   - Los programas de transferencia posteriores al destino se han ejecutado correctamente.

Procedimiento

- Especifique el programa que desea ejecutar utilizando una de las opciones siguientes:

  Utilizar una tarea Apache Ant

  Utilizar una de las tareas `fte:filecopy`, `fte:filemove` y `fte:call` de Ant para iniciar un programa. Utilizando una tarea de Ant, puede especificar un programa en cualquiera de los cinco escenarios utilizando los elementos anidados `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrc` y `fte:command`. Para obtener más información, consulte Elementos anidados de invocación de programa.

  Editar el mensaje de solicitud de transferencia de archivos

  Puede editar el XML generado por una solicitud de transferencia. Con este método, puede ejecutar un programa en cualquiera de los cinco escenarios, añadiendo los elementos **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall** y **managedCall** al archivo XML. A continuación, utilice este archivo XML modificado como la definición de transferencia para una nueva solicitud de transferencia de archivos, por ejemplo con el parámetro **-td** del mandato **fteCreateTransfer**. Para obtener más información, consulte Ejemplos de mensajes de solicitud para llamadas de agente MFT .

  Utilizar el mandato `fteCreateTransfer`

  Puede utilizar el mandato **fteCreateTransfer** para especificar programas que desea iniciar. Puede utilizar el mandato para especificar programas para ejecutar en los cuatro primeros escenarios, como parte de una solicitud de transferencia, pero no puede iniciar una llamada gestionada. Para obtener información sobre los parámetros a utilizar, consulte **fteCreateTransfer**: iniciar una nueva transferencia de archivos. Para ver ejemplos de

cómo utilizar este mandato, consulte Ejemplos de utilización de fteCreateTransfer para iniciar programas.

Referencia relacionada

Propiedad commandPath de MFT

Llamadas gestionadas

Los agentes de Managed File Transfer (MFT) se utilizan normalmente para transferir archivos o mensajes. Estos se conocen como *transferencias gestionadas*. Los agentes también se pueden utilizar para ejecutar mandatos, scripts o JCL sin necesidad de transferir archivos o mensajes. Esta prestación se conoce como *Llamadas gestionadas*.

Las solicitudes de llamadas gestionadas se pueden enviar a un agente de varias maneras:

- Utilizando la tarea fte: call Ant.
- Configuración de un supervisor de recursos con un XML de tarea que ejecuta un mandato o un script. Consulte Configuración de tareas de supervisión para iniciar mandatos y scripts para obtener más información.
- Colocar directamente un mensaje XML en la cola de mandatos del agente. Consulte Formato de mensaje de solicitud de transferencia de archivos para obtener más detalles sobre el esquema XML de llamada gestionada.

Para las llamadas gestionadas, el directorio que contiene el mandato o script que se está ejecutando debe especificarse en la propiedad de agente **commandPath**.

Las llamadas gestionadas no pueden ejecutar mandatos o scripts que se encuentren en directorios que no se hayan especificado en el **commandPath**del agente. Esto es para asegurarse de que el agente no ejecuta ningún código malicioso.

Importante: Para asegurarse de que este es el caso, de forma predeterminada, cuando especifica **commandPath**:

- El agente configura cualquier recinto de seguridad de agente existente cuando se inicia, de forma que todos los directorios **commandPath** se añaden automáticamente a la lista de directorios a los que se ha denegado el acceso para una transferencia.
- Los recintos de seguridad de usuario existentes se actualizan cuando se inicia el agente para que todos los directorios **commandPath** (y sus subdirectorios) se añadan como elementos `<exclude>` a los elementos `<read>` y `<write>` .
- Si el agente no está configurado para utilizar un recinto de seguridad de agente o un recinto de seguridad de usuario, se crea un nuevo recinto de seguridad de agente cuando se inicia el agente que tiene los directorios **commandPath** especificados como directorios denegados.

Además, también puede habilitar la comprobación de autorización en un agente para asegurarse de que sólo los usuarios autorizados pueden enviar solicitudes de llamadas gestionadas. Para obtener más información al respecto, consulte Restricción de autorizaciones de usuario en acciones de agente de MFT.

El mandato, script o JCL invocado como parte de una llamada gestionada se ejecuta como un proceso externo, supervisado por el agente. Cuando el proceso sale, la llamada gestionada se completa y el código de retorno del proceso se pone a disposición del agente o del script Ant que ha invocado la tarea **fte:call** Ant .

Si la tarea **fte:call** Ant ha iniciado la llamada gestionada, el script Ant puede comprobar el valor del código de retorno para determinar si la llamada gestionada ha sido satisfactoria o no.

Para todos los demás tipos de llamadas gestionadas, puede especificar qué valores de código de retorno deben utilizarse para indicar que la llamada gestionada se ha completado correctamente. El agente compara el código de retorno del proceso con estos códigos de retorno cuando finaliza el proceso externo.

Nota: Puesto que las llamadas gestionadas se ejecutan como procesos externos, no se pueden cancelar una vez que se han iniciado.

Llamadas gestionadas y ranuras de transferencia de origen

Un agente contiene un número de ranuras de transferencia de origen, tal como se especifica en la propiedad de agente **maxSourceTransfers**, que se describe en Propiedades avanzadas del agente: Límite de transferencia.

Siempre que se ejecuta una llamada gestionada o una transferencia gestionada, ocupan una ranura de transferencia de origen. La ranura se libera cuando se completa la llamada gestionada o la transferencia gestionada.

Si todas las ranuras de transferencia de origen están en uso cuando un agente recibe una nueva llamada gestionada o una solicitud de transferencia gestionada, el agente pone la solicitud en cola hasta que una ranura pasa a estar disponible.

Si una llamada gestionada inicia una transferencia gestionada (por ejemplo, si una llamada gestionada ejecuta un script Ant y dicho script Ant utiliza la tarea fte: filecopy o fte: filemove para transferir un archivo), se necesitan dos ranuras de transferencia de origen:

- Uno para la transferencia gestionada
- Uno para la llamada gestionada

En esta situación, es importante tener en cuenta que si la transferencia gestionada tarda mucho tiempo en completarse o entra en recuperación, las dos ranuras de transferencia de origen estarán ocupadas hasta que se complete la transferencia gestionada, se cancele o se agote el tiempo de espera debido a un **transferRecoveryTimeout**. Consulte Conceptos de tiempo de espera de recuperación de transferencia para obtener detalles sobre **transferRecoveryTimeout**. Esto puede limitar potencialmente el número de otras transferencias gestionadas o llamadas gestionadas que el agente puede procesar.

Debido a esto, debe considerar el diseño de una llamada gestionada para asegurarse de que no ocupa ranuras de transferencia de origen durante un largo periodo de tiempo.

Utilización de REST API con llamadas gestionadas

V 9.3.0 V 9.3.0

Los verbos HTTP GET y HTTP POST están soportados para habilitar llamadas gestionadas y solo funcionan en la versión 3 de REST API.

Otros verbos, por ejemplo, HTTP DELETE y HTTP UPDATE no están soportados y devuelven el código de error HTTP 405 si intenta utilizarlos.

Atención: Una vez enviada una llamada gestionada, no se puede cancelar utilizando REST API.

Utilización de Apache Ant con MFT

Managed File Transfer proporciona tareas que puede utilizar para integrar la función de transferencia de archivos en la herramienta Apache Ant.

Puede utilizar el mandato **fteAnt** para ejecutar tareas Ant en un entorno Managed File Transfer que ya ha configurado. Puede utilizar las tareas Ant de transferencia de archivos de los scripts Ant para coordinar las operaciones de transferencia de archivos complejas desde un lenguaje de script interpretado.

Para obtener más información sobre Apache Ant, consulte la página web del proyecto Apache Ant : https://ant.apache.org/

Conceptos relacionados

“Cómo empezar a utilizar scripts Ant con MFT” en la página 1254
El uso de scripts Ant con Managed File Transfer le permite coordinar operaciones de transferencia de archivos complejas desde un lenguaje de script interpretado.

fteAnt: ejecutar tareas Ant en MFT

Referencia relacionada

"Tareas de ejemplo Ant para MFT" en la página 1255
Existe una serie de scripts de ejemplo Ant que se proporcionan con la información de Managed File Transfer. Estos ejemplos se encuentran en el directorio *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de ejemplo contiene un destino `init`, edite las propiedades establecidas en el destino `init` para ejecutar estos scripts con la configuración.

Cómo empezar a utilizar scripts Ant con MFT

El uso de scripts Ant con Managed File Transfer le permite coordinar operaciones de transferencia de archivos complejas desde un lenguaje de script interpretado.

Scripts Ant

Los scripts Ant (o archivos de compilación) son documentos XML que definen uno o varios destinos. Estos destinos contienen elementos de tarea que se van a ejecutar. Managed File Transfer proporciona tareas que puede utilizar para integrar la función de transferencia de archivos en Apache Ant. Para obtener más información sobre los scripts Ant , consulte la página web del proyecto Apache Ant : https://ant.apache.org/

Los ejemplos de scripts Ant que utilizan tareas Managed File Transfer se proporcionan con la instalación del producto en el directorio *MQ_INSTALLATION_PATH*/mqft/samples/fteant

En los agentes de puente de protocolo, los scripts Ant se ejecutan en el sistema de agente de puente de protocolo. Estos scripts Ant no tienen acceso directo a los archivos en el servidor FTP o SFTP.

Espacio de nombres

Se utiliza un espacio de nombres para diferenciar las tareas Ant de transferencia de archivos de otras tareas Ant que podrían compartir el mismo nombre. Defina el espacio de nombres en la etiqueta del proyecto del script Ant.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

El atributo `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` indica a Ant que busque las definiciones de tareas con el prefijo `fte` en la biblioteca `com.ibm.wmqfte.ant.taskdefs`.

No es necesario utilizar `fte` como prefijo de espacio de nombres; puede utilizar cualquier valor. El prefijo de espacio de nombres `fte` se utiliza en todos los ejemplos y los scripts Ant de ejemplo.

Ejecución de scripts Ant

Para ejecutar los scripts Ant que contienen las tareas Ant de transferencia de archivos, utilice el mandato **fteAnt**. Por ejemplo:

```
fteAnt -file ant_script_location/ant_script_name
```

Para obtener más información, consulte **fteAnt**: ejecutar tareas Ant en MFT.

Códigos de retorno

Las tareas Ant de transferencia de archivos devuelven los mismos códigos de retorno que los mandatos Managed File Transfer. Para obtener más información, consulte Códigos de retornos para MFT.

Referencia relacionada

fteAnt: ejecutar tareas Ant en MFT

“Tareas de ejemplo Ant para MFT” en la página 1255

Existe una serie de scripts de ejemplo Ant que se proporcionan con la información de Managed File Transfer. Estos ejemplos se encuentran en el directorio *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de ejemplo contiene un destino `init`, edite las propiedades establecidas en el destino `init` para ejecutar estos scripts con la configuración.

Tareas de ejemplo Ant para MFT

Existe una serie de scripts de ejemplo Ant que se proporcionan con la información de Managed File Transfer. Estos ejemplos se encuentran en el directorio *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Cada script de ejemplo contiene un destino `init`, edite las propiedades establecidas en el destino `init` para ejecutar estos scripts con la configuración.

email

El ejemplo email ilustra cómo utilizar las tareas Ant para transferir un archivo y enviar un correo electrónico a una dirección de correo electrónico especificada, si la transferencia falla. El script comprueba que los agentes de origen y de destino están activos y que pueden procesar transferencias utilizando la tarea de Managed File Transfer ping. Si ambos agentes están activos, el script utiliza la tarea Managed File Transfer fte:filecopy para transferir un archivos entre los agentes de origen y de destino, sin suprimir el archivo original. Si la transferencia falla, el script envía un correo electrónico que contiene información sobre la anomalía utilizando la tarea de Ant email estándar.

hub

El ejemplo hub se compone de dos scripts: `hubcopy.xml` y `hubprocess.xml`. El script `hubcopy.xml` muestra cómo puede utilizar los scripts Ant para crear topologías de estilo 'hub and spoke'. En este ejemplo, se transfieren dos archivos de agentes que se ejecutan en máquinas radiales a un agente que se ejecuta en la máquina de eje. Ambos archivos se transfieren a la vez, y cuando las transferencias se completan, el script `hubprocess.xml` Ant se ejecuta en la máquina de concentrador para procesar los archivos. Si ambos archivos se transfieren correctamente, el script Ant concatena el contenido de los archivos. Si los archivos no se transfieren correctamente, el script Ant hace limpieza suprimiendo todos los datos de archivo que se han transferido. Para que este ejemplo funcione correctamente, debe poner el script `hubprocess.xml` en la vía de acceso de mandatos del agente de eje. Para obtener más información sobre cómo establecer la vía de acceso del mandato de un agente, consulte Propiedad commandPath MFT.

librarytransfer (sólo en la plataforma IBM i)

IBM i

IBM i El librarytransfer de ejemplo ilustra cómo utilizar las tareas Ant para transferir una biblioteca de IBM i en un sistema IBM i a un segundo sistema IBM i.

IBM i El librarytransfer de ejemplo utiliza el soporte de archivos de salvar nativo en IBM i con tareas Ant predefinidas disponibles en Managed File Transfer para transferir objetos de biblioteca nativa entre dos sistemas IBM i. El ejemplo utiliza un elemento < presrc> anidado en una tarea Managed File Transfer filecopy para invocar un script ejecutable `librarysave.sh` que guarda la biblioteca solicitada en el sistema del agente de origen en un archivo de salvar temporal. La tarea ant filecopy mueve el archivo de salvar al sistema del agente de destino donde se utiliza un elemento anidado < postdst> para invocar el script ejecutable `libraryrestore.sh` para restaurar la biblioteca guardada en el archivo de salvar al sistema de destino.

IBM i Antes de ejecutar este ejemplo debe completar algunas tareas de configuración, descritas en el archivo `librarytransfer.xml`. También debe tener un entorno de Managed File Transfer funcional en dos máquinas IBM i. La configuración debe constar de un agente de origen ejecutándose

en la primera máquina IBM i y un agente de destino ejecutándose en la segunda máquina IBM i. Los dos agentes deben poder comunicarse entre sí.

IBM i El ejemplo librarytransfer consta de los tres archivos siguientes:

- `librarytransfer.xml`
- `librarysave.sh` (script ejecutable < presrc>)
- `libraryrestore.sh` (script ejecutable < postdst>)

Los archivos de ejemplo se encuentran en el siguiente directorio: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Para ejecutar este ejemplo el usuario debe completar los pasos siguientes:

1. Inicie una sesión de Qshell. En una ventana de mandatos de IBM i, escriba: STRQSH
2. Vaya al directorio `bin` de la siguiente manera:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Después de completar la configuración necesaria, ejecute el ejemplo ejecutando el mandato siguiente:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer (sólo en la plataforma IBM i)

IBM i El ejemplo de physicalfiletransfer ilustra cómo utilizar las tareas Ant para transferir un archivo de base de datos o físico de origen de una biblioteca en un sistema IBM i a una biblioteca en un segundo sistema IBM i.

IBM i El ejemplo de physicalfiletransfer utiliza el soporte de archivo de salvar nativo en IBM i con las tareas Ant predefinidas disponibles en Managed File Transfer para transferir los archivos de base de datos y físicos completos entre dos sistemas IBM i. El ejemplo utiliza un elemento anidado < presrc> dentro de una tarea de Managed File Transfer filecopy para invocar un script ejecutable `physicalfilesave.sh` para guardar el archivo físico o de base de datos de origen solicitado de una biblioteca en el sistema del agente de origen en un archivo de salvar temporal. La tarea ant filecopy mueve el archivo de salvar al sistema del agente de destino donde se utiliza un elemento anidado < postdst> para invocar el script ejecutable `physicalfilerestore.sh` y, a continuación, restaura el objeto de archivo dentro del archivo de salvar en una biblioteca especificada en el sistema de destino.

IBM i Antes de ejecutar este ejemplo debe completar algunas tareas de configuración, tal como se describe en el archivo `physicalfiletransfer.xml`. También debe tener un entorno de Managed File Transfer funcional en dos sistemas IBM i. La configuración debe consistir en un agente de origen que se ejecute en el primer sistema IBM i y un agente de destino que se ejecute en el segundo sistema IBM i. Los dos agentes deben poder comunicarse entre sí.

IBM i El ejemplo physicalfiletransfer consta de los tres archivos siguientes:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (script ejecutable < presrc>)
- `physicalfilerestore.sh` (script ejecutable < postdst>)

Los archivos de ejemplo se encuentran en el siguiente directorio: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Para ejecutar este ejemplo el usuario debe completar los pasos siguientes:

1. Inicie una sesión de Qshell. En una ventana de mandatos de IBM i, escriba: STRQSH

2. Vaya al directorio `bin` de la siguiente manera:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Después de completar la configuración necesaria, ejecute el ejemplo ejecutando el mandato siguiente:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

tiempo de espera

El ejemplo timeout demuestra cómo utilizar tareas Ant para intentar una transferencia de archivos y para cancelarla si tarda más tiempo que un valor de tiempo de espera excedido especificado. El script inicia una transferencia de archivos utilizando la tarea Managed File Transfer fte:filecopy. El resultado de esta transferencia se aplaza. El script utiliza la Managed File Transfertarea fte:awaitoutcome Ant para esperar durante un número de segundos especificado a que se complete la transferencia. Si la transferencia no se completa en el periodo de tiempo especificado, se utiliza la Managed File Transfertarea fte:cancel Ant para cancelar la transferencia de archivos.

vsamtransfer

z/OS

z/OS El ejemplo de vsamtransfer ilustra cómo utilizar las tareas Ant para transferir de un conjunto de datos VSAM a otro conjunto de datos utilizando Managed File Transfer. Managed File Transfer actualmente no da soporte a la transferencia de conjuntos de datos VSAM. El script de ejemplo descarga los registros de datos VSAM en un conjunto de datos secuencial utilizando los `presrc` Elementos anidados de invocación de programa para llamar al archivo ejecutable `datasetcopy.sh`. El script utiliza la tarea Managed File Transfer fte:filemove para transferir el conjunto de datos secuenciales del agente de origen al agente de destino. A continuación, el script utiliza los `postdst` Elementos anidados de invocación de programa para llamar al script `loadvsam.jcl` . Este script JCL carga los registros del conjunto de datos transferidos a un conjunto de datos VSAM. Este ejemplo utiliza JCL para la llamada de destino para mostrar esta opción de lenguaje. También se puede lograr el mismo resultado utilizando en su lugar un segundo script de shell.

z/OS Este ejemplo no requiere que los conjuntos de datos de origen y destino sean VSAM. El ejemplo funciona para cualquier conjunto de datos si los conjuntos de datos de origen y de destino son del mismo tipo.

z/OS Para que este ejemplo funcione correctamente, debe poner el script `datasetcopy.sh` en la vía de acceso de mandatos del agente de origen y el script `loadvsam.jcl` en la vía de acceso de mandatos del agente de destino. Para obtener más información sobre cómo establecer la vía de acceso del mandato de un agente, consulte Propiedad commandPath MFT.

zip

El ejemplo zip se compone de dos scripts: `zip.xml` y `zipfiles.xml`. El ejemplo muestra cómo utilizar el `presrc` elemento anidado dentro de la tarea fte: filemove de Managed File Transfer para ejecutar un script Ant antes de realizar una operación de traslado de transferencia de archivos. El script `zipfiles.xml` invocado por el elemento anidado `presrc` en el script `zip.xml` comprime el contenido de un directorio. El script `zip.xml` transfiere el archivo comprimido. Este ejemplo requiere que el script `zipfiles.xml` Ant esté presente en la vía de acceso de mandatos del agente de origen. Esto se debe a que el script `zipfiles.xml` Ant contiene el destino utilizado para comprimir el contenido del directorio en el agente de origen. Para obtener más información sobre cómo establecer la vía de acceso del mandato de un agente, consulte Propiedad commandPath MFT.

Conceptos relacionados

“Cómo empezar a utilizar scripts Ant con MFT” en la página 1254

El uso de scripts Ant con Managed File Transfer le permite coordinar operaciones de transferencia de archivos complejas desde un lenguaje de script interpretado.

Referencia relacionada

fteAnt: ejecutar tareas Ant en MFT

Personalización de MFT con salidas de usuario

Puede personalizar las características de Managed File Transfer utilizando sus propios programas, conocidos como rutinas de salida de usuario.

Importante: Cualquier código dentro de una salida de usuario no está soportado por IBM, y cualquier problema con ese código debe ser investigado inicialmente por la empresa o el proveedor que ha proporcionado la salida.

Managed File Transfer proporciona puntos en el código en los que Managed File Transfer puede pasar el control a un programa que el usuario ha escrito (una rutina de salida de usuario). Estos puntos se conocen como puntos de salida de usuario. Managed File Transfer puede posteriormente retomar el control cuando el programa ha finalizado el trabajo. No es necesario que utilice ninguna de las salidas de usuario, pero son útiles si desea ampliar y personalizar la función del sistema Managed File Transfer para satisfacer sus necesidades específicas.

Existen dos puntos durante el proceso de transferencias de archivos en los que puede invocar una salida de usuario en el sistema de origen y dos puntos durante el proceso de transferencia de archivos en los que puede invocar una salida de usuario en el sistema de destino. En la tabla siguiente se resumen cada uno de los puntos de salida de usuario y la interfaz Java que debe implementar para utilizar estos puntos de salida.

Tabla 187. Resumen de puntos de salida del lado de origen y del lado de destino así como las interfaces Java

Punto de salida	**Interfaz Java que se implementa**
Puntos de salida del lado de origen:	
Antes de que se inicie toda la transferencia de archivos	Interfaz SourceTransferStartExit.java
Después de que finalice la transferencia de archivos completa	Interfaz SourceTransferEndExit.java
Puntos de salida en el destino:	
Antes de que se inicie toda la transferencia de archivos	Interfaz DestinationTransferStartExit.java
Después de que finalice la transferencia de archivos completa	Interfaz DestinationTransferEndExit.java

Las salidas de usuario se invocan en el orden siguiente:

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

Los cambios efectuados en las salidas SourceTransferStartExit y DestinationTransferStartExit se propagan como entrada en salidas posteriores. Por ejemplo, si la salida SourceTransferStartExit modifica los metadatos de transferencia, los cambios se reflejan en los metadatos de transferencia de entrada a otras salidas.

Las salidas de usuario y las llamadas de programa se invocan en el orden siguiente:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Notas:

1. El **DestinationTransferEndExits** sólo se ejecuta cuando finaliza la transferencia, ya sea de forma satisfactoria o parcial.
2. El **postDestinationCall** sólo se ejecuta cuando finaliza la transferencia, ya sea de forma satisfactoria o parcial.
3. El **SourceTransferEndExits** se ejecuta para transferencias satisfactorias, parcialmente satisfactorias o anómalas.
4. **postSourceCall** sólo se llama si:
   - La transferencia no se ha cancelado.
   - Hay un resultado satisfactorio o parcialmente satisfactorio.
   - Los programas de transferencia posteriores al destino se han ejecutado correctamente.

Compilación de salida de usuario

Las interfaces para crear una salida de usuario están contenidas en *MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar. Debe incluir este archivo .jar en la vía de acceso de clases cuando cree la salida. Para ejecutar la salida, extraiga la salida como un archivo .jar y coloque este archivo .jar en un directorio tal como se describe en la siguiente sección.

Ubicaciones de salida de usuario

Puede almacenar las rutinas de salida en dos ubicaciones posibles:

- El directorio exits. Existe un directorio exits en cada directorio de agente. Por ejemplo: var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits
- Puede establecer que la propiedad exitClassPath especifique una ubicación alternativa. Si existen clases de salida en el directorio exits y en las vías de acceso de clases establecidas en exitClassPath, tienen prioridad las clases del directorio exits, lo que significa que si existen clases en ambas ubicaciones con el mismo nombre, tienen prioridad las clases del directorio exits.

Configuración de un agente para utilizar salidas de usuario

Hay cuatro propiedades de agente que se pueden establecer para especificar las salidas de usuario que un agente invoca. Estas propiedades de agente son sourceTransferStartExitClasses, sourceTransferEndExitClasses, destinationTransferStartExitClasses y destinationTransferEndExitClasses. Para obtener más información sobre cómo utilizar estas propiedades, consulte Propiedades de agente MFT para salidas de usuario.

Ejecución de salidas de usuario en agentes de puente de protocolo

Cuando el agente de origen invoca la salida, pasa la salida de una lista de los elementos de origen para la transferencia. Para los agentes normales, se trata de una lista de nombres de archivo completos. Puesto que los archivos deben ser locales (o accesibles a través de un montaje), la salida es capaz de acceder al mismo y de cifrarlo.

Sin embargo, para un agente de puente de protocolo, las entradas de la lista tienen el formato siguiente:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Para cada entrada de la lista, la salida se debe conectar primero al servidor de archivos (utilizando los protocolos FTP. FTPS o SFTP), descargar el archivo, cifrarlo localmente y, después, volver a cargar el archivo cifrado en el servidor de archivos.

Ejecución de salidas de usuario en agentes de puente Connect:Direct

No puede ejecutar salidas de usuario en agentes de puente Connect:Direct.

Conceptos relacionados

"Salidas de usuario de origen y destino de MFT" en la página 1260

Metadatos para salidas de usuario de MFT

Interfaces de Java para salidas de usuario de MFT

Referencia relacionada

"Habilitación de la depuración remota para salidas de usuario de MFT" en la página 1265

Mientras desarrolla salidas de usuario, es posible que desee utilizar un depurador para ayudarle a localizar problemas con el código.

"Salida de usuario de transferencia de origen de MFT" en la página 1265

"Ejemplo de salida de usuario de credenciales de puente de protocolo" en la página 1266

Salidas de usuario del supervisor de recursos de MFT

Propiedades de agente MFT para salidas de usuario

Salidas de usuario de origen y destino de MFT

Separadores de directorio

Los separadores de directorio en las especificaciones de archivo de origen siempre se representan utilizando caracteres de barra inclinada (/), independientemente de cómo haya especificado separadores de directorio en el mandato **fteCreateTransfer** o en IBM MQ Explorer. Debe tenerlo en cuenta cuando escriba una salida. Por ejemplo, si desea comprobar si el siguiente archivo de origen existe: `c:\a\b.txt` y ha especificado este archivo de origen mediante el mandato **fteCreateTransfer** o IBM MQ Explorer, tenga presente que el nombre de archivo está almacenado actualmente como: `c:/a/b.txt`. Por lo tanto, si busca la serie original de `c:\a\b.txt`, no encontrará ninguna coincidencia.

Puntos de salida del lado del origen

Antes de que se inicie toda la transferencia de archivos

El agente de origen invoca esta salida cuando una solicitud de transferencia es la siguiente en la lista de transferencias pendientes y la transferencia está a punto de empezar.

El envío de archivos por etapas a un directorio en el que el agente tiene acceso de lectura/grabación utilizando un mandato externo o la redenominación de los archivos en el sistema de destino son ejemplos de uso de este punto de salida.

Transfiera los siguientes argumentos a esta salida:

- Nombre del agente de origen
- Nombre del agente de destino
- Metadatos del entorno
- Metadatos de transferencia
- Especificaciones de archivo (incluidos los metadatos de archivo)

Los datos devueltos por esta salida son los siguientes:

- Metadatos de transferencia actualizados. Las entradas se pueden añadir, modificar y suprimir.

- Lista actualizada de especificaciones de archivos, que consiste en pares de nombres de archivo de destino y nombres de archivo de origen. Las entradas se pueden añadir, modificar y suprimir.
- Indicador que especifica si hay que continuar la transferencia
- Serie para insertar en el registro de transferencias.

Implemente la interfaz SourceTransferStartExit.java para llamar al código de salida de usuario en este punto de salida.

Después de que finalice la transferencia de archivos completa
El agente de origen invoca esta salida después de que se complete la transferencia de archivos completa.

Un ejemplo de uso de este punto de salida es realizar algunas tareas de terminación, tales como enviar un correo electrónico o un mensaje de IBM MQ para indicar que ha terminado la transferencia.

Transfiera los siguientes argumentos a esta salida:

- Resultado de salida de transferencia
- Nombre del agente de origen
- Nombre del agente de destino
- Metadatos del entorno
- Metadatos de transferencia
- Resultados del archivo

Los datos devueltos por esta salida son los siguientes:

- Serie actualizada para insertar en el registro de transferencias.

Implemente la interfaz SourceTransferEndExit.java para llamar al código de salida de usuario en este punto de salida.

Puntos de salida del lado del destino

Antes de que se inicie toda la transferencia de archivos
Un uso de ejemplo de este punto de salida es validar los permisos en el destino.

Transfiera los siguientes argumentos a esta salida:

- Nombre del agente de origen
- Nombre del agente de destino
- Metadatos del entorno
- Metadatos de transferencia
- Especificaciones de archivo

Los datos devueltos por esta salida son los siguientes:

- Conjunto actualizado de nombres de archivo de destino. Las entradas se pueden modificar, pero no se pueden añadir o suprimir.
- Indicador que especifica si hay que continuar la transferencia
- Serie para insertar en el registro de transferencias.

Implemente la interfaz DestinationTransferStartExit.java para llamar al código de salida de usuario en este punto de salida.

Después de que finalice la transferencia de archivos completa
Un uso de ejemplo de esta salida de usuario es iniciar un proceso por lotes que utiliza los archivos transferidos o enviar un correo electrónico si la transferencia ha fallado.

Transfiera los siguientes argumentos a esta salida:

- Resultado de salida de transferencia

- Nombre del agente de origen
- Nombre del agente de destino
- Metadatos del entorno
- Metadatos de transferencia
- Resultados del archivo

Los datos devueltos por esta salida son los siguientes:

- Serie actualizada para insertar en el registro de transferencias.

Implemente la interfaz DestinationTransferEndExit.java para llamar al código de salida de usuario en este punto de salida.

Conceptos relacionados

Interfaces de Java para salidas de usuario de MFT

Referencia relacionada

“Habilitación de la depuración remota para salidas de usuario de MFT” en la página 1265
Mientras desarrolla salidas de usuario, es posible que desee utilizar un depurador para ayudarle a localizar problemas con el código.

“Salida de usuario de transferencia de origen de MFT” en la página 1265

Salidas de usuario del supervisor de recursos de MFT

Utilización de salidas de usuario de E/S de transferencia de MFT

Puede utilizar salidas de usuario de E/S de transferencia de Managed File Transfer para configurar código personalizado para realizar el trabajo de E/S del sistema de archivos subyacente para transferencias de Managed File Transfer.

Generalmente, para las transferencias de MFT, un agente selecciona uno de los proveedores de E/S incorporados para interactuar con los sistemas de archivos adecuados para la transferencia. Los proveedores de E/S incorporados dan soporte a los siguientes tipos de sistema de archivos:

- Sistemas de archivos regulares de tipo UNIX y de tipo Windows
- z/OS Conjuntos de datos secuenciales y particionados de z/OS (solamente en z/OS)
- IBM i Archivos de salvar nativos de IBM i (solamente en IBM i)
- Colas de IBM MQ
- Servidores de protocolo FTP y SFTP remotos (sólo para agentes de puente de protocolo)
- Nodos Connect:Direct remotos (sólo para agentes de puente de Connect:Direct)

Para los sistemas de archivos que no están soportados, o cuando necesite un comportamiento de E/S personalizado, puede escribir una salida de usuario de E/S de transferencia.

Las salidas de usuario de E/S de transferencia utilizan la infraestructura existente para salidas de usuario. Sin embargo, estas salidas de usuario de E/S de transferencia difieren de otras salidas de usuario, ya que se accede a su función varias veces durante la transferencia para cada archivo.

Utilice la propiedad de agente IOExitClasses (en el archivo `agent.properties`) para especificar las clases de salida de E/S que se han de cargar. Separe cada clase de salida con una coma, por ejemplo:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Las interfaces Java de las salidas de usuario de E/S de transferencia son las siguientes:

IOExit
El punto de entrada principal que se utiliza para determinar si se utiliza la salida de E/S. Esta instancia es responsable de crear instancias de IOExitPath.

Sólo necesita especificar la interfaz de salida de E/S IOExit para la propiedad de agente IOExitClasses.

IOExitPath
Representa una interfaz abstracta; por ejemplo, un contenedor de datos o un comodín que representa un conjunto de contenedores de datos. No puede crear una instancia de clase que implemente esta interfaz. La interfaz permite examinar la vía de acceso y listar las vías de acceso derivadas. Las interfaces IOExitResourcePath e IOExitWildcardPath amplían IOExitPath.

IOExitChannel
Permite leer datos o grabar datos en un recurso IOExitPath.

IOExitRecordChannel
Amplía la interfaz IOExitChannel para recursos IOExitPath orientados a registros, lo que permite leer datos o grabar datos en un recurso IOExitPath en múltiplos de registros.

IOExitLock
Representa un bloqueo en un recurso IOExitPath para acceso compartido o exclusivo.

z/OS **IOExitRecordResourcePath**
Extiende la interfaz IOExitResourcePath para representar un contenedor de datos de un archivo orientado a registro; por ejemplo, un conjunto de datos z/OS. Puede utilizar la interfaz para localizar datos y para crear instancias de IOExitRecordChannel para operaciones de lectura o grabación.

IOExitResourcePath
Amplía la interfaz IOExitPath para representar un contenedor de datos; por ejemplo, un archivo o directorio. Puede utilizar la interfaz para localizar datos. Si la interfaz representa un directorio, puede utilizar el método listPaths para devolver una lista de vías de acceso.

IOExitWildcardPath
Amplía la interfaz IOExitPath para representar una vía de acceso que denota un comodín. Puede utilizar esta interfaz para que coincida con varias IOExitResourcePaths.

IOExitProperties
Especifica las propiedades que determinan cómo Managed File Transfer maneja IOExitPath para determinados aspectos de E/S. Por ejemplo, si se deben utilizar archivos intermedios o si se debe volver a leer un recurso desde el principio si se reinicia una transferencia.

Conceptos relacionados
“Personalización de MFT con salidas de usuario” en la página 1258
Puede personalizar las características de Managed File Transfer utilizando sus propios programas, conocidos como rutinas de salida de usuario.

Referencia relacionada
Interfaz IOExit.java
Interfaz IOExitChannel.java
Interfaz IOExitLock.java
Interfaz IOExitPath.java
Interfaz IOExitProperties.java
Interfaz IOExitRecordChannel.java
z/OS Interfaz IOExitRecordResourcePath.java
Interfaz IOExitResourcePath.java
Interfaz IOExitWildcardPath.java
El archivo MFT `agent.properties`

IBM i Ejemplo de MFT salidas de usuario de IBM i

Managed File Transfer proporciona salidas de usuario de ejemplo específicas de IBM i con la instalación. Los ejemplos se encuentran en los directorios *MQMFT_install_dir*/samples/ioexit-IBMi y *MQMFT_install_dir*/samples/userexit-IBMi.

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit

La salida de usuario de ejemplo com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit transfiere archivos del sistema de archivos QDLS en IBM i. Después de instalar la salida, las transferencia a los archivos que empiezan con /QDLS utilizan automáticamente la salida.

Para instalar esta salida, complete los pasos siguientes:

1. Copie el archivo com.ibm.wmqfte.samples.ibmi.ioexits.jar del directorio *WMQFTE_install_dir*/samples/ioexit-IBMi en el directorio exits del agente.
2. Añada com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit a la propiedad IOExitClasses.
3. Reinicie el agente.

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit

La salida de usuario de ejemplo com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit se comporta como un supervisor de archivos MFT y transfiere automáticamente los miembros de archivos físicos de una biblioteca de IBM i.

Para ejecutar esta salida, especifique un valor para el campo de metadatos "library.qsys.monitor" (utilizando el parámetro **-md**, por ejemplo). Este parámetro toma una vía de acceso de estilo IFS a un miembro de archivo y puede contener caracteres comodín de archivo y miembro. Por ejemplo, /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.MBR.

Esta salida de ejemplo también tiene un campo opcional de metadatos "naming.scheme.qsys.monitor", que puede utilizar para determinar el esquema de denominación que se utiliza durante la transferencia. De forma predeterminada, este campo se establece en "unix,", lo que hace que el archivo de destino se llame F00.MBR. También puede especificar el valor "ibmi" para utilizar IBM i FTP FILE.MEMBER , por ejemplo, /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR se transfiere como BAR.BAZ.

Para instalar esta salida, complete los pasos siguientes:

1. Copie el archivo com.ibm.wmqfte.samples.ibmi.userexits.jar del directorio *WMQFTE_install_dir*/samples/userexit-IBMi en el directorio exits del agente.
2. Añada com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit a la propiedad sourceTransferStartExitClasses en el archivo agent.properties.
3. Reinicie el agente.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

La salida de usuario de ejemplo com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit suprime un objeto de archivo vacío cuando el miembro de archivo de origen se suprime como parte de la transferencia. Puesto que los objetos de archivo de IBM i pueden potencialmente contener muchos miembros, MFT trata los objetos de archivo como directorios. Por lo tanto, no puede realizar una operación de traslado en un objeto de archivo utilizando MFT; las operaciones de traslado sólo se soportan a nivel de miembro. Por lo tanto, cuando se realiza una operación de traslado en un miembro, el archivo que ahora está vacío se deja atrás. Utilice esta salida de ejemplo si desea suprimir estos archivos vacíos como parte de la solicitud de transferencia.

Si especifica "true" para los metadatos "empty.file.delete" y transferir un FTEFileMember, la salida de ejemplo suprime el archivo padre si el archivo está vacío.

Para instalar esta salida, complete los pasos siguientes:

1. Copie el archivo com.ibm.wmqfte.samples.ibmi.userexits.jar de *WMQFTE_install_dir*/samples/userexit-IBMi en el directorio exits del agente.

2. Añada com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit a la propiedad sourceTransferStartExitClasses en el archivo `agent.properties`.
3. Reinicie el agente.

Referencia relacionada

"Utilización de salidas de usuario de E/S de transferencia de MFT" en la página 1262
Puede utilizar salidas de usuario de E/S de transferencia de Managed File Transfer para configurar código personalizado para realizar el trabajo de E/S del sistema de archivos subyacente para transferencias de Managed File Transfer.

Propiedades de agente MFT para salidas de usuario

Habilitación de la depuración remota para salidas de usuario de MFT

Mientras desarrolla salidas de usuario, es posible que desee utilizar un depurador para ayudarle a localizar problemas con el código.

Puesto que las salidas se ejecutan dentro de la máquina virtual Java que ejecuta el agente, no puede utilizar el soporte de depuración directo que se incluye habitualmente en un entorno de desarrollo integrado. No obstante, puede habilitar la depuración remota de la JVM y luego conectar un depurador remoto adecuado.

Para habilitar la depuración remota, utilice los parámetros JVM estándar **-Xdebug** y **-Xrunjdwp**. Estas propiedades se pasan a la JVM que ejecuta el agente mediante la variable de entorno **BFG_JVM_PROPERTIES** . Por ejemplo, en AIX and Linux los mandatos siguientes inician el agente y hacen que la JVM esté a la escucha de conexiones del depurador en el puerto TCP 8765.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

El agente no se inicia hasta que se conecta el depurador. Use el mandato **set** en Windows en lugar del mandato **export**.

También puede utilizar otros métodos de comunicación entre el depurador y la JVM. Por ejemplo, la JVM puede abrir la conexión al depurador en lugar de al revés, o bien puede utilizar la memoria compartida en vez de TCP. Consulte la documentación de Java Platform Debugger Architecture si dese más detalles.

Debe utilizar el parámetro **-F** (en primer plano) cuando inicie el agente en modalidad de depuración remota.

Utilización del depurador de Eclipse

Los pasos siguientes se aplican a la prestación de depuración remota del entorno de desarrollo de Eclipse. También puede utilizar otros depuradores remotos que son compatibles con JPDA.

1. Pulse **Ejecutar** > **Abrir diálogo de depuración** (o **Ejecutar** > **Configuraciones de depuración** o **Ejecutar** > **Diálogo de depuración** en función de la versión de Eclipse).
2. Efectúe una doble pulsación en **Aplicación Java remota** en la lista de tipos de configuración para crear una configuración de depuración.
3. Complete los campos de configuración y guarde la configuración de depuración. Si ya ha iniciado la JVM de agente en la modalidad de depuración, puede conectarse ahora a la JVM.

Salida de usuario de transferencia de origen de MFT

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
```

```
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

Ejemplo de salida de usuario de credenciales de puente de protocolo

Para obtener información sobre cómo utilizar esta salida de usuario de ejemplo, consulte Correlación de credenciales para un servidor de archivos utilizando clases de salida.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
```

```
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            }
            catch (FileNotFoundException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                 System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                initialisationResult = false;
            }
            catch (IOException ex) {
                System.err.println("Error initializing SampleCredentialExit.");
                 System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                initialisationResult = false;
            }
            finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    }
                    catch (IOException ex) {
                        System.err.println("Error initializing SampleCredentialExit.");
                         System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of mqUserId to server credentials from the properties
                final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                 while ( propertyNames.hasMoreElements()) {
                    final Object name = propertyNames.nextElement();
                    if (name instanceof String ) {
                        final String mqUserId = ((String)name).trim();
                         // Get the value and split into serverUserId and serverPassword
                       final String value = mappingProperties.getProperty(mqUserId);
                       final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                        String serverUserId = "";
                        String serverPassword = "";
                        if (valueTokenizer.hasMoreTokens()) {
                            serverUserId = valueTokenizer.nextToken().trim();
                        }
                        if (valueTokenizer.hasMoreTokens()) {
                            serverPassword = valueTokenizer.nextToken().trim();
                        }
                        // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                        // Insert the credentials into the map
                        credentialsMap.put(mqUserId, credentials);
                    }
                }

           }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
```

```
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

Ejemplo de salida de usuario de propiedades de puente de protocolo

Para obtener información sobre cómo utilizar esta salida de usuario de ejemplo, consulte ProtocolBridgePropertiesExit2: búsqueda de propiedades del servidor de archivos de protocolo

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
```

```
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
```

```
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

Control de MFT colocando mensajes en la cola de mandatos de agente

Puede escribir una aplicación que controle Managed File Transfer colocando mensajes en colas de mandatos de agente.

Puede poner un mensaje en la cola de mandatos de un agente para solicitar que el agente realice una de las acciones siguientes:

- Crear una transferencia de archivos
- Crear una transferencia de archivos planificada
- Cancelar una transferencia de archivos
- Cancelar una transferencia de archivos planificada
- Llamar a un mandato
- Crear un supervisor
- Suprimir un supervisor
- Devolver un mensaje de sondeo para indicar que el agente está activo

Para solicitar que el agente realice una de estas acciones, el mensaje debe estar en un formato XML que siga uno de los esquemas siguientes:

`FileTransfer.xsd`
Los mensajes con este formato se pueden utilizar para crear una transferencia de archivos o una transferencia de archivos planificada, para llamar a un mandato o para cancelar una transferencia de archivos o una transferencia de archivos planificada. Para obtener más información, consulte Formato de mensaje de solicitud para transferencia de archivos.

`Monitor.xsd`
Los mensajes con este formato se pueden utilizar para crear o suprimir un supervisor de recursos. Para obtener más información, consulte Formatos de mensajes de solicitud de supervisor MFT.

`PingAgent.xsd`
Los mensajes con este formato se pueden utilizar para sondear un agente a fin de comprobar que está activo. Para obtener más información, consulte Formato de mensaje de solicitud para sondear un agente MFT

El agente devuelve una respuesta a los mensajes de solicitud. El mensaje de respuesta se coloca en una cola de respuestas que está definida en el mensaje de solicitud. El mensaje de solicitud tiene un formato XML definido por el esquema siguiente:

`Reply.xsd`
Para obtener más información, consulte Formato de mensaje de respuesta de agente MFT .

Desarrollo de aplicaciones para MQ Telemetry

Las aplicaciones de telemetría integran dispositivos de detección y control con otras fuentes de información disponibles en Internet y en las empresas.

Desarrolle aplicaciones para MQ Telemetry utilizando patrones de diseño, ejemplos preparados, programas de ejemplo, conceptos de programación e información de referencia.

Conceptos relacionados

MQ Telemetry

Casos de uso de telemetría

Tareas relacionadas

Instalación del MQ Telemetry

Administración de MQ Telemetry

Resolución de problemas de MQ Telemetry

Referencia relacionada

Referencia de MQ Telemetry

Programas de ejemplo de IBM MQ Telemetry Transport

Se proporcionan scripts de ejemplo que funcionan con una aplicación cliente IBM MQ Telemetry Transport v3 de ejemplo (`mqttv3app.jar`). Para IBM MQ 8.0.0 y posteriores, la aplicación cliente de ejemplo ya no se incluye en MQ Telemetry. Formaba parte de IBM Messaging Telemetry Clients SupportPac(ya no disponible). Las aplicaciones de ejemplo similares siguen estando disponibles gratuitamente en Eclipse Paho y MQTT.org.

Para obtener la información y las descargas más recientes, consulte los recursos siguientes:

- El proyecto Eclipse Paho y MQTT.org, tienen descargas gratuitas de los últimos clientes de telemetría y ejemplos para un rango de lenguajes de programación. Utilice estos sitios para ayudarle a desarrollar programas de ejemplo para la publicación y suscripción de IBM MQ Telemetry Transport y para añadir características de seguridad.
- IBM Messaging Telemetry Clients SupportPac ya no está disponible para la descarga. Si tiene una copia descargada anteriormente, esta contiene lo siguiente:
  - La versión MA9B de IBM Messaging Telemetry Clients SupportPac incluía una aplicación de ejemplo compilada (`mqttv3app.jar`) y una biblioteca de cliente asociada (`mqttv3.jar`). Se han proporcionado en los directorios siguientes:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - En la versión MA9C de este SupportPac, se ha eliminado el directorio y el contenido de `/SDK/`:
    - Sólo se ha proporcionado el origen de la aplicación de ejemplo (`mqttv3app.jar`). Estaba en este directorio:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La biblioteca de cliente compilada aún se proporcionaba. Estaba en este directorio:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Si todavía tiene una copia de IBM Messaging Telemetry Clients SupportPac(ya no disponible), se proporciona información sobre la instalación y ejecución de la aplicación de ejemplo en Verificación de la instalación de MQ Telemetry utilizando la línea de mandatos.

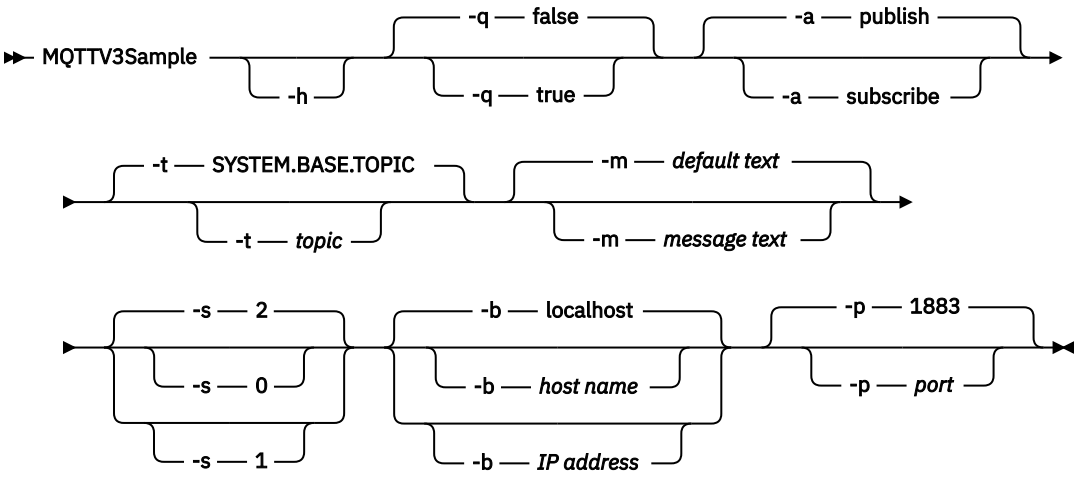
Programa MQTTV3Sample

Información de referencia sobre los parámetros y la sintaxis de ejemplo para el programa `MQTTV3Sample`.

Finalidad

El programa `MQTTV3Sample` se puede utilizar para publicar un mensaje y suscribirse a un tema. Para obtener más información sobre cómo obtener este programa de ejemplo, consulte “Programas de ejemplo de IBM MQ Telemetry Transport” en la página 1273.

MQTTV3Sample syntax



Parámetros

-h
Imprimir el texto de ayuda y salir

-q
Establecer la modalidad silenciosa (-q), en lugar de utilizar la modalidad predeterminada, que es false.

-a
Establecer publish o subscribe, en lugar de aceptar la acción predeterminada, que es publish.

-t
Publicar o suscribirse a un tema, en lugar de publicar o suscribirse al tema predeterminado

-m
Publicar un mensaje de texto, en lugar de enviar el texto de publicación predeterminado: "Hello from an MQTT v3 application".

-s
Establecer la calidad de servicio (QoS), en lugar de utilizar la calidad de servicio predeterminada, que es 2.

-b
Conectar con el nombre de host o dirección IP especificado, en lugar de conectar con el nombre de host predeterminado, que es localhost.

-p
Utilizar el puerto especificado, en lugar de utilizar el puerto predeterminado, que es 1883.

Ejecute el programa MQTTV3Sample

Para suscribirse a un tema en Windows, utilice este mandato:

```
run MQTTV3Sample -a subscribe
```

Para publicar un mensaje en Windows, utilice este mandato:

```
run MQTTV3Sample
```

Conceptos sobre la programación del cliente de MQTT

Los conceptos que se describen en esta sección le ayudan a conocer las bibliotecas de cliente de MQTT protocol. Estos conceptos complementan la documentación de API que acompaña a las bibliotecas de cliente.

Para obtener la información y las descargas más recientes, consulte los recursos siguientes:

- El proyecto Eclipse Paho y MQTT.org, tienen descargas gratuitas de los últimos clientes de telemetría y ejemplos para un rango de lenguajes de programación. Utilice estos sitios para ayudarle a desarrollar programas de ejemplo para la publicación y suscripción de IBM MQ Telemetry Transport y para añadir características de seguridad.
- IBM Messaging Telemetry Clients SupportPac ya no está disponible para la descarga. Si tiene una copia descargada anteriormente, esta contiene lo siguiente:
  - La versión MA9B de IBM Messaging Telemetry Clients SupportPac incluía una aplicación de ejemplo compilada (`mqttv3app.jar`) y una biblioteca de cliente asociada (`mqttv3.jar`). Se han proporcionado en los directorios siguientes:
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - En la versión MA9C de este SupportPac, se ha eliminado el directorio y el contenido de `/SDK/`:
    - Sólo se ha proporcionado el origen de la aplicación de ejemplo (`mqttv3app.jar`). Estaba en este directorio:

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La biblioteca de cliente compilada aún se proporcionaba. Estaba en este directorio:

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Para desarrollar y ejecutar un cliente MQTT debe copiar o instalar estos recursos en el dispositivo cliente. No es necesario que instale un cliente aparte en tiempo de ejecución.

Las condiciones de licencia para los clientes van asociadas al servidor al que conecta los clientes.

Las bibliotecas de cliente de MQTT son implementaciones de referencia de MQTT protocol. Puede implementar sus propios clientes en los distintos lenguajes adecuados a sus distintas plataformas de dispositivos. Consulte Formato y protocolo de IBM MQ Telemetry Transport.

La documentación de la API no hace ninguna suposición respecto a qué servidor de MQTT está conectado el cliente. El comportamiento del cliente puede diferir ligeramente cuando está conectado a servidores diferentes. Las descripciones que siguen a continuación describen el comportamiento del cliente cuando está conectado al servicio de telemetría de IBM MQ.

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT

El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Devoluciones de llamadas

Nota: Consulte el sitio web de Eclipse Paho para ver los últimos cambios en `MqttCallback`. Por ejemplo, `MqttCallback` se define como una interfaz en la versión Paho del cliente, y la clase `MqttAsyncClient` de Paho proporciona métodos asíncronos.

La interfaz `MqttCallback` tiene tres métodos de devolución de llamada:

connectionLost(java.lang.Throwable cause)

Se invoca `connectionLost` cuando la conexión se cierra por un error de comunicaciones. También se invoca si el servidor cierra la conexión como resultado de un error en el servidor después de establecerse la conexión. Los errores del servidor se registran en el registro de errores del gestor de colas. El servidor cierra la conexión con el cliente y el cliente invoca `MqttCallback.connectionLost`.

Los únicos errores remotos que se emiten como excepciones en la misma hebra que la aplicación cliente son las excepciones de `MqttClient.connect`. Los errores que el servidor detecta una vez establecida la conexión se notifican al método callback `MqttCallback.connectionLost` como `throwables`.

Los errores de servidor habituales dan como resultado la invocación de `connectionLost` son los errores de autorización. Por ejemplo, el servidor de telemetría intenta publicar en un tema en nombre de un cliente que no tiene autorización para publicar en el tema. Todo lo que produzca la devolución de un código de condición `MQCC_FAIL` al servidor de telemetría puede dar como resultado el cierre de la conexión.

deliveryComplete(IMqttDeliveryToken token)

El cliente de MQTT llama a `deliveryComplete` para devolver una señal de entrega a la aplicación cliente; consulte “Señales de entrega” en la página 1282. Cuando se utiliza una señal de entrega, la devolución de llamada puede acceder al mensaje publicado mediante el método `token.getMessage`.

Cuando la devolución de llamada de la aplicación devuelve el control al cliente de MQTT después de ser invocada por el método `deliveryComplete`, la entrega se ha completado. Hasta que no se completa la entrega, la clase de persistencia retiene los mensajes con `QoS` 1 ó 2.

La llamada `deliveryComplete` constituye un punto de sincronización entre la aplicación y la clase de persistencia. Nunca se llama al método `deliveryComplete` dos veces para el mismo mensaje.

Cuando la devolución de llamada de aplicación vuelve de `deliveryComplete` al cliente MQTT , el cliente llama a `MqttClientPersistence.remove` para mensajes con `QoS` 1 o 2. `MqttClientPersistence.remove` suprime la copia almacenada localmente del mensaje publicado.

Desde una perspectiva de proceso de transacción, la llamada a `deliveryComplete` es una transacción de una sola fase que confirma la entrega. Si falla el proceso durante la devolución de llamada, al reiniciar el cliente se invoca de nuevo `MqttClientPersistence.remove` para suprimir la copia local del mensaje publicado. La devolución de llamada no se invoca de nuevo. Si utiliza la devolución de llamada para almacenar un registro de los mensajes entregados, puede sincronizar el registro con el cliente MQTT. Si desea almacenar un registro de forma segura, actualice el registro en la clase `MqttClientPersistence`.

La hebra principal de la aplicación y el cliente MQTT hacen referencia a la señal de entrega y al mensaje. El cliente MQTT anula la referencia al objeto `MqttMessage` cuando se ha completado la entrega, y al objeto de señal de entrega cuando el cliente se desconecta. El objeto `MqttMessage` puede ser eliminado por el recolector de basura una vez completada la entrega si la aplicación cliente desreferencia el objeto. La señal de entrega puede ser eliminada por el recolector de basura una vez que se ha desconectado la sesión.

Puede obtener los atributos `IMqttDeliveryToken` y `MqttMessage` después de que se haya publicado un mensaje. Si intenta definir cualquier atributo `MqttMessage` después de que el mensaje se haya publicado, no puede definirse el resultado.

El cliente de MQTT continúa procesando los confirmaciones de entrega si el cliente se vuelve a conectar a la sesión anterior con el mismo `ClientIdentifier`; consulte “Limpiar sesiones” en la página 1279. La aplicación cliente de MQTT debe establecer `MqttClient.CleanSession` en `false` para la sesión anterior, y establecerlo en `false` en la nueva sesión. El cliente de MQTT crea nuevas señales de entrega y objetos de mensaje en la nueva sesión para las entregas pendientes. Recupera los objetos utilizando la clase `MqttClientPersistence`. Si el cliente de la aplicación todavía tiene referencias a señales de entrega y mensajes antiguos, elimine las referencias a ellos. La devolución de llamada de aplicación se invoca en la nueva sesión para todas las entregas iniciadas en la sesión anterior y completadas en la sesión actual.

La devolución de llamada de aplicación se invoca después de que el cliente de aplicación se conecte, cuando se completa una entrega pendiente. Antes de que se conecte el cliente de la aplicación, puede recuperar entregas pendientes con el método `MqttClient.getPendingDeliveryTokens`.

Observe que la aplicación cliente originalmente creó el objeto de mensaje que se publica y su matriz de bytes de carga. El cliente de MQTT hace referencia a estos objetos. El objeto de mensaje devuelto por la señal de entrega en el método `token.getMessage` no es necesariamente el mismo objeto de mensaje que creó el cliente. Si una nueva instancia de cliente MQTT vuelve a crear la señal de entrega, la clase `MqttClientPersistence` vuelve a crear el objeto `MqttMessage`. Por razones de coherencia, `token.getMessage` devuelve `null` si `token.isCompleted` está definido en `true`, independientemente de si el objeto de mensaje lo creó el cliente de la aplicación o la clase `MqttClientPersistence`.

`messageArrived(String topic, MqttMessage message)`

Se llama a `messageArrived` cuando llega una publicación para el cliente que coincide con un tema de suscripción. `topic` es el tema de publicación, no el filtro de suscripción. Pueden ser diferentes si el filtro contiene comodines.

Si el tema coincide con varias suscripciones creadas por el cliente, este recibe varias copias de la publicación. Si un cliente publica en un tema al que también está suscrito, recibe una copia de su propia publicación.

Si se envía un mensaje con QoS igual a 1 o 2, el mensaje se almacena en la clase `MqttClientPersistence` antes de que el cliente MQTT llame a `messageArrived`. `messageArrived` se comporta como `deliveryComplete`: es invocado una sola vez para una publicación y `MqttClientPersistence.remove` elimina la copia local de la publicación cuando `messageArrived` devuelve el control al cliente MQTT. El cliente MQTT elimina sus referencias al tema y mensaje cuando `messageArrived` devuelve el control al cliente MQTT. Los objetos de tema y mensaje son eliminados por el recolector de basura si el cliente de aplicación no ha retenido una referencia a los objetos.

Sincronización de devoluciones de llamada, generación de hebras y aplicaciones cliente

El cliente MQTT llama a un método de devolución de llamada en una hebra distinta de la hebra de aplicación principal. La aplicación cliente no crea una hebra para la devolución de llamada; es creada por el cliente MQTT.

El cliente MQTT sincroniza los métodos de devolución de llamada. Se ejecuta una sola instancia del método de devolución de llamada cada vez. La sincronización permite actualizar fácilmente un objeto que cuenta las publicaciones que se han entregado. Se ejecuta una sola instancia de `MqttCallback.deliveryComplete` cada vez, por lo que es seguro actualizar el recuento sin realizar más sincronización. Es el mismo caso que cuando llega una sola publicación cada vez. El código del método `messageArrived` puede actualizar un objeto sin sincronizarlo. Si va a hacer una referencia al recuento o al objeto que se está actualizando en otra hebra, sincronice el recuento o el objeto.

La señal de entrega proporciona un mecanismo de sincronización entre la hebra de aplicación principal y la entrega de una publicación. El método `token.waitForCompletion` espera hasta que se completa una publicación específica o hasta que finaliza un tiempo de espera opcional. Puede utilizar `token.waitForCompletion` de la forma siguiente para procesar una publicación cada vez.

Para sincronizar con el método `MqttCallback.deliveryComplete`. Sólo cuando `MqttCallback.deliveryComplete` vuelve al cliente MQTT reanuda `token.waitForCompletion`. Utilizando este mecanismo puede sincronizar el código de ejecución en `MqttCallback.deliveryComplete` antes de que se ejecute el código en la hebra de la aplicación principal.

¿Qué ocurre si desea publicar sin esperar que se entregue cada publicación, pero desea confirmación cuando se hayan entregado todas las publicaciones? Si realiza la publicación en una única hebra, la última publicación que se envía es también la última publicación que se entrega.

Sincronización de solicitudes enviadas al servidor

Tabla 188 en la página 1278 describe los métodos en el cliente MQTT Java que envía una solicitud al servidor. A menos que el cliente de aplicaciones establezca un tiempo de espera indefinido, el cliente nunca esperará de forma indefinida la respuesta del servidor. Si el cliente se bloquea, es un problema de programación de aplicación o un defecto en el cliente de MQTT.

Tabla 188. Comportamiento de la sincronización de métodos que tienen como resultado la creación de solicitudes para el servidor

Método	Sincronización	Intervalo de tiempo de espera
`MqttClient.Connect`	Espera a que se establezca una conexión con el servidor.	El valor predeterminado es 30 segundos, o el valor que haya establecido un parámetro, y después emite una excepción.
`MqttClient.Disconnect`	Espera a que el cliente MQTT finalice el trabajo que debe hacer y que la sesión TCP/IP se desconecte.	
`MqttClient.Subscribe`	Espera a que finalice el método Subscribe o UnSubscribe.	
`MqttClient.UnSubscribe`		
`MqttClient.Publish`	Devuelve inmediatamente el control a la hebra de aplicación después de pasar la solicitud al cliente MQTT.	Ninguna.
`IMqttDeliveryToken.waitForCompletion`	Espera a que se devuelva la señal de entrega.	Indefinido, o el valor que esté establecido como parámetro.

Conceptos relacionados

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Limpiar sesiones

El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Cuando se conecta una aplicación cliente de MQTT utilizando el método `MqttClient.connect`, el cliente identifica la conexión utilizando el identificador de cliente y la dirección del servidor. El servidor comprueba si se ha guardado información de sesión procedente de una conexión anterior con el servidor. Si todavía existe una sesión anterior y se ha especificado `cleanSession=true`, se borra la información de la sesión anterior en el cliente y en el servidor. Si `cleanSession=false`, se reanuda la sesión anterior. Si no existe ninguna sesión anterior, se inicia una nueva.

Nota: El administrador de IBM MQ puede forzar el cierre de una sesión abierta y suprimir toda la información de la sesión. Si el cliente reabre la sesión con la opción `cleanSession=false`, se inicia una sesión nueva.

Publicaciones

Si utiliza el valor predeterminado `MqttConnectOptions`, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar con el cliente, se eliminan todas las entregas de publicaciones pendientes para el cliente cuando éste se conecta.

El valor de limpiar sesión no afecta a las publicaciones enviadas con `QoS=0`. Para `QoS=1` y `QoS=2`, la utilización de `cleanSession=true` puede provocar la pérdida de una publicación.

Suscripciones

Si utiliza el `MqttConnectOptions`predeterminado, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar el cliente, las suscripciones antiguas para el cliente se eliminan cuando se conecta el cliente. Las suscripciones nuevas que el cliente realice durante la sesión se eliminan cuando se desconecta.

Si establece `MqttConnectOptions.cleanSession` en `false` antes de conectarse, las suscripciones que crea el cliente se añaden a todas las suscripciones que existían para el cliente antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otro modo de entender la forma en la que el atributo `cleanSession` afecta a las suscripciones es pensar en él como un atributo modal. Con la modalidad predeterminada, `cleanSession=true`, el cliente crea suscripciones y recibe publicaciones sólo dentro del ámbito de la sesión. En la modalidad alternativa, `cleanSession=false`, las suscripciones son duraderas. El cliente puede conectarse y desconectarse y las suscripciones permanecen activas. Cuando el cliente vuelve a conectarse, recibe las publicaciones que no se hayan entregado. Mientras está conectado, puede modificar el conjunto de suscripciones que estén activas en su nombre.

Debe establecer la modalidad `cleanSession` antes de conectarse; la modalidad dura toda la sesión. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de uso de `cleanSession=false` a `cleanSession=true`, se descartarán todas las suscripciones anteriores para el cliente y las publicaciones que no se hayan recibido.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían

al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Identificador de cliente

El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

El identificador de cliente se utiliza en la administración de un sistema MQTT. Con cientos de miles de clientes potenciales que administrar, es necesario poder identificar un cliente concreto rápidamente. Por ejemplo, supongamos que un dispositivo ha funcionado mal y se le notifica, quizás mediante un cliente que hace un llamada a un centro de atención al cliente. El cliente necesita poder identificar el dispositivo y debe poder correlacionar esa identificación con el servidor que normalmente está conectado al cliente.

Cuando examina conexiones de clientes MQTT, todas las conexiones están etiquetadas con el identificador de cliente. Para ayudarle a decidir la mejor manera de correlacionar este identificador con el dispositivo y el servidor, hágase las siguientes preguntas:

- ¿Es conveniente mantener y utilizar una base de datos que correlacione cada dispositivo con un identificador de cliente y con un servidor?
- ¿Podría el nombre del dispositivo identificar el servidor al que está conectado?
- ¿Necesita una tabla de búsqueda que correlacione un identificador de cliente con un dispositivo físico?
- ¿El identificador de cliente identifica a un dispositivo específico, a un usuario o a una aplicación que se ejecuta en el cliente?
- Si un cliente sustituye un dispositivo defectuoso por uno nuevo, ¿tiene el nuevo dispositivo el mismo identificador que el dispositivo antiguo o asigna un nuevo identificador? (Si cambia un dispositivo físico y mantiene el mismo identificador, las publicaciones pendientes y las suscripciones activas se transfieren automáticamente al nuevo dispositivo.)

También necesita un sistema para asegurarse de que los identificadores de cliente son exclusivos y debe tener un proceso fiable para establecer el identificador en el cliente. Si el dispositivo cliente es una "caja negra", sin interfaz de usuario, puede fabricar el dispositivo con un identificador de cliente, o puede tener un proceso de instalación y configuración de software que configure el dispositivo antes de activarlo.

Para mantener el identificador corto y exclusivo, puede crear un identificador de cliente a partir de la dirección MAC del dispositivo de 48 bits. Si el tamaño de transmisión no es un problema crítico, puede utilizar los 17 bytes restantes para facilitar la administración de la dirección.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Señales de entrega

Cuando un cliente publica en un tema se crea una nueva señal de entrega. Utilice la señal de entrega para supervisar la entrega de una publicación, para bloquear la aplicación cliente hasta que se complete la entrega.

La señal es un objeto `MqttDeliveryToken`. Se crea al llamar al método MqttTopic.publish() y la retiene el cliente MQTT hasta que la sesión de cliente se desconecta y se completa la entrega.

La señal se utiliza normalmente para comprobar si se ha completado la entrega. Puede bloquear la aplicación cliente hasta que se complete la entrega utilizando la señal devuelta para llamar a `token.waitForCompletion`. Como alternativa, puede proporcionar un controlador `MqttCallBack`. Cuando el cliente MQTT recibe todos los acuses de recibo que espera como parte de la entrega de una publicación, llama al método `MqttCallBack.deliveryComplete`, pasando la señal de entrega como parámetro.

Hasta que se complete la entrega, puede examinar la publicación mediante la señal de entrega devuelta llamando a `token.getMessage`.

Entregas completadas
La finalización de las entregas es asíncrona, y depende de la calidad de servicio asociada a la publicación.

Como máximo una vez

QoS=0
La entrega se completa inmediatamente tras la devolución por parte de `MqttTopic.publish`. Se llama inmediatamente a `MqttCallback.deliveryComplete`.

Al menos una vez

QoS=1
La entrega se completa cuando se recibe en la publicación un acuse de recibo proceden del gestor de colas. Cuando se recibe el acuse de recibo, se llama inmediatamente a `MqttCallback.deliveryComplete`. Es posible que el mensaje se entregue más de una vez antes de que se llame a `MqttCallback.deliveryComplete`, si la comunicación es lenta o no es fiable.

Exactamente una vez

QoS=2
La entrega se completa cuando el cliente recibe un mensaje de finalización indicando que la publicación se ha publicado para los suscriptores. Se llama a `MqttCallback.deliveryComplete` tan pronto como se recibe el mensaje de publicación. No espera al mensaje que indica la finalización.

En casos raros, la aplicación cliente puede no devolver el control al cliente MQTT después de ejecutar normalmente `MqttCallback.deliveryComplete`. Sabrá que la entrega se ha completado porque se ha invocado `MqttCallback.deliveryComplete`. Si el cliente reinicia la misma sesión, no se llama de nuevo a `MqttCallback.deliveryComplete`.

Entregas incompletas

Si la entrega no se ha completado después de que la sesión de cliente se desconecte, puede conectar el cliente de nuevo y completar la entrega. Sólo puede completar la entrega de un mensaje si éste se publica en una sesión con el atributo `MqttConnectionOptions` establecido en `false`.

Cree el cliente utilizando el mismo identificador de cliente y la dirección de servidor, y conéctelo entonces estableciendo el atributo `cleanSession` `MqttConnectionOptions` de nuevo en `false`. Si establece `cleanSession` en `true`, se descartan las señales de entrega pendientes.

Puede comprobar si existe alguna entrega pendiente llamando a `MqttClient.getPendingDeliveryTokens`. Puede llamar a `MqttClient.getPendingDeliveryTokens` antes de conectar el cliente.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Publicación de última voluntad y testamento

Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Cree un tema para la última voluntad y testamento. Puede crear un tema como, por ejemplo, `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Configure una "última voluntad y testamento" mediante el método `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Considere la posibilidad de crear una indicación de la hora en el mensaje `lastWillPayload`. Incluya otra información de cliente que ayude a identificar el cliente y las circunstancias de la conexión. Pase el objeto `MqttConnectionOptions` al constructor `MqttClient`.

Establezca `lastWillQos` en 1 o 2 para hacer que el mensaje sea persistente en IBM MQ y asegurar la entrega. Para que se conserve la información de la última conexión perdida, establezca `lastWillRetained` en `true`.

La publicación "última voluntad y testamento" se envía a los suscriptores si la conexión finaliza de forma inesperada. Se envía si la conexión finaliza sin que el cliente llame al método `MqttClient.disconnect`.

Para supervisar las conexiones, complemente la publicación "última voluntad y testamento" con otras publicaciones para registrar las conexiones y desconexiones programadas.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Persistencia de mensajes en clientes MQTT

Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente

o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

En MQTT, la persistencia de mensajes tiene dos aspectos: cómo se transfiere el mensaje y si se pone en cola en IBM MQ como un mensaje persistente.

1. El cliente MQTT asocia la persistencia de mensaje con la calidad de servicio. Dependiendo de la calidad de servicio que elija para un mensaje, el mensaje pasa a ser persistente. La persistencia de mensaje es necesaria para implementar la calidad de servicio necesaria.

   Si especifica "como máximo una vez", `QoS=0`, el cliente rechaza el mensaje tan pronto como se publica. Si existe algún error en las comunicaciones en sentido ascendente, el mensaje no se vuelve a enviar. Incluso aunque el cliente permanezca activo, el mensaje no se vuelve a enviar. El comportamiento de los mensajes con `QoS=0` es el mismo que para lo mensajes no persistentes rápidos de IBM MQ.

   Si un cliente publica un mensaje con `QoS` establecido en a 1 ó 2, el mensaje pasa a ser persistente. El mensaje se almacena localmente y sólo se descarta del cliente cuando ya no es necesario garantizar "al menos una vez", `QoS=1`o "exactamente una vez", `QoS=2`, la entrega.
2. Si un mensaje se marca como `QoS` 1 o 2, se pone en cola en IBM MQ como un mensaje persistente. Si se marca como `QoS=0`, se pone en cola en IBM MQ como un mensaje no persistente. En IBM MQ, los mensajes no persistentes se transfieren entre gestores de colas "exactamente una vez", a menos que el canal de mensajes tenga el atributo `NPMSPEED` establecido en `FAST`.

Una publicación persistente se almacena en el cliente hasta que es recibida por una aplicación cliente. Para `QoS=2`, la publicación se descarta del cliente cuando la devolución de llamada de la aplicación devuelve el control. Para `QoS=1` la aplicación puede volver a recibir la publicación si se produce un error. Para `QoS=0`, la devolución de llamada recibe la publicación no más de una vez. Es posible que no reciba la publicación si existe un error o si el cliente se desconecta en el momento de la publicación.

Cuando se suscribe a un tema, puede reducir la `QoS` con la que el suscriptor recibe mensajes para que la calidad de servicio sea conforme con los recursos de persistencia del suscriptor. Las publicaciones que se crean con una `QoS` mayor se envían con la `QoS` más alta que el suscriptor solicitó.

Almacenamiento de mensajes

La implementación del almacenamiento de datos en dispositivos pequeños varía mucho. El método para guardar temporalmente mensajes persistentes en un espacio de almacenamiento gestionado por el cliente MQTT puede ser demasiado lento o exigir demasiado espacio de almacenamiento. En los dispositivos móviles, el sistema operativo para dispositivos móviles puede proporcionar un servicio de almacenamiento que es ideal para los mensajes de MQTT.

Para proporcionar flexibilidad y tener en cuenta las limitaciones de los dispositivos pequeños, el cliente MQTT tiene dos interfaces de persistencia. Las interfaces definen las operaciones que intervienen en el almacenamiento de mensajes persistentes. Las interfaces se describen en la documentación de la API para el MQTT client for Java. Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación del cliente MQTT. Puede implementar las interfaces para que se adapten a un dispositivo. El cliente de MQTT que se ejecuta en Java SE tiene una implementación predeterminada de las interfaces que almacenan mensajes persistentes en el sistema de archivos. Esta implementación utiliza el paquete `java.io`.

Clases de persistencia

`MqttClientPersistence`
: Esta clase pasa una instancia de la implementación de `MqttClientPersistence` al cliente MQTT como parámetro del constructor `MqttClient`. Si omite el parámetro `MqttClientPersistence` del constructor `MqttClient`, el cliente MQTT almacena los mensajes persistentes utilizando la clase `MqttDefaultFilePersistence`.

`MqttPersistable`
: `MqttClientPersistence` obtiene y coloca objetos `MqttPersistable` mediante una clave de almacenamiento. Debe proporcionar una implementación de `MqttPersistable`

así como la implementación de `MqttClientPersistence` si no está utilizando `MqttDefaultFilePersistence`.

`MqttDefaultFilePersistence`

El cliente MQTT proporciona la clase `MqttDefaultFilePersistence`. Si crea una instancia de `MqttDefaultFilePersistence` en su aplicación cliente, puede proporcionar el directorio para almacenar mensajes persistentes como parámetro del constructor `MqttDefaultFilePersistence`.

De forma alternativa, el cliente de MQTT puede crear una instancia de `MqttDefaultFilePersistence` y colocar los archivos en el siguiente directorio predeterminado:

`client identifier` `-tcp` *`hostname portnumber`*

Los caracteres siguientes se eliminan de la serie de nombre de directorio:

`"\", "\\", "/", ":"y" "`

La vía de acceso al directorio es el valor de la propiedad del sistema `rcp.data`; si no se ha establecido `rcp.data` , la vía de acceso es el valor de la propiedad del sistema `usr.data`, donde

- `rcp.data` es una propiedad asociada a la instalación de OSGi o Eclipse Rich Client Platform (RCP).
- `usr.data` es el directorio donde se emitió el mandato Java por el que se ha iniciado la aplicación.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Publicaciones

Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Un `MqttMessage` tiene una matriz de bytes como carga útil. Intente mantener los mensajes lo más pequeños posible. La longitud máxima de mensaje permitida por el MQTT protocol es 250 MB.

Generalmente, un programa cliente MQTT utiliza `java.lang.String` o `java.lang.StringBuffer` para manipular el contenido de los mensajes. Para su comodidad, `MqttMessage` utiliza un método `toString` para convertir la carga en una serie. Para crear una carga útil de matriz de bytes a partir de un `java.lang.String` o `java.lang.StringBuffer`, utilice el método `getBytes`.

El método `getBytes` convierte una serie al conjunto de caracteres predeterminado de la plataforma. El conjunto de caracteres predeterminado es normalmente `UTF-8`. La publicaciones de MQTT que contienen sólo texto, generalmente están en `UTF-8`. Utilice el método `getBytes("UTF8")` para anular temporalmente el conjunto de caracteres predeterminado.

En IBM MQ, una publicación de MQTT se recibe como un mensaje de `jms-bytes`. El mensaje incluye una carpeta `MQRFH2` que contiene una carpeta `<mqtt>` y una carpeta `<mqps>`. La carpeta `<mqtt>` contiene `clientId`, `msgId` y `qos`, pero su contenido podría cambiar en el futuro.

Un `MqttMessage` tiene tres atributos adicionales: la calidad de servicio, un indicador de retención y un indicador de duplicado. El indicador de duplicado se define sólo si la calidad de servicio es "al menos una vez" o "exactamente una vez". Si el mensaje se ha enviado anteriormente y el cliente MQTT no lo ha reconocido lo suficientemente rápido, el mensaje se vuelve a enviar con el atributo de duplicado definido en `true`.

Publicación

Para crear una publicación en una aplicación cliente MQTT, cree un `MqttMessage`. Defina la carga útil, la calidad de servicio y un indicador de retención para el mensaje e invoque el método `MqttTopic.publish(MqttMessage message)`; se devuelve `MqttDeliveryToken` y la finalización de la publicación es asíncrona.

De forma alternativa, el cliente de MQTT puede crear un objeto de mensaje temporal a partir de los parámetros del método `MqttTopic.publish(byte [] payload, int qos, boolean retained)` cuando crea una publicación.

Si la calidad de servicio de la publicación es "al menos una vez" o "exactamente una vez", `QoS=1` o `QoS=2`, el cliente MQTT invoca la interfaz `MqttClientPersistence`. Llama a `MqttClientPersistence` para almacenar el mensaje antes de devolver una señal de entrega a la aplicación.

La aplicación puede elegir bloquear hasta que el mensaje se haya entregado al servidor, utilizando el método `MqttDeliveryToken.waitForCompletion`. Como alternativa, la aplicación puede continuar sin realizar bloqueo. Si desea comprobar que las publicaciones se entregan, sin realizar un bloqueo,

registre una instancia de una clase de devolución de llamada que implemente `MqttCallback` con el cliente MQTT. El cliente MQTT llama al método `MqttCallback.deliveryComplete` tan pronto como se entrega la publicación. Según la calidad de servicio, la entrega puede ser casi inmediata con `QoS=0` o puede tardar un poco con `QoS=2`.

Utilice el método `MqttDeliveryToken.isComplete` para averiguar si la entrega se ha completado. Mientras el valor de `MqttDeliveryToken.isComplete` sea `false`, puede llamar a `MqttDeliveryToken.getMessage` para obtener el contenido del mensaje. Si el resultado al llamar a `MqttDeliveryToken.isComplete` es `true`, se ha rechazado el mensaje y al llamar a `MqttDeliveryToken.getMessage` debe aparecer una excepción de puntero nulo. No existe sincronización generada entre `MqttDeliveryToken.getMessage` y `MqttDeliveryToken.isComplete`.

Si el cliente se desconecta antes de recibir todas las señales de entrega pendientes, con una nueva instancia de cliente se puede consultar las señales de entrega pendientes antes de conectarse. Hasta que el cliente se conecte, no se completa ninguna nueva entrega y es seguro llamar a `MqttDeliveryToken.getMessage`. Utilice el método `MqttDeliveryToken.getMessage` para averiguar qué publicaciones no se han entregad aún. Las señales de entrega pendientes se descartan si se conecta con `MqttConnectOptions.cleanSession` definido en `true`, su valor predeterminado.

Suscripción

Un gestor de colas es responsable de crear publicaciones para enviar a un suscriptor de MQTT . El gestor de colas comprueba si el filtro de temas de una suscripción creada por un cliente MQTT coincide con la serie de tema de una publicación. La coincidencia puede ser exacta o incluir comodines. Antes de reenviar la publicación al suscriptor con el gestor de colas, este comprueba los atributos de temas asociados a la publicación. Se sigue el procedimiento de búsqueda que se describe en Suscripción utilizando una serie de tema que contiene caracteres comodín para identificar si un objeto de tema administrativo otorga al usuario la autorización para suscribirse.

Cuando el cliente MQTT recibe una publicación con una calidad de servicio de "al menos una vez", llama al método `MqttCallback.messageArrived` para procesar la publicación. Si la calidad de servicio de la publicación es "exactamente una vez", `QoS=2`, el cliente MQTT llama a la interfaz `MqttClientPersistence` para almacenar el mensaje cuando se reciba. A continuación llama a `MqttCallback.messageArrived`.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento

Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Calidades de servicio proporcionadas por un cliente MQTT

Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

La calidad de servicio de una publicación es un atributo de `MqttMessage`. Es establecido por el método `MqttMessage.setQos`.

El método `MqttClient.subscribe` puede reducir la calidad de servicio que se aplica a las publicaciones enviadas a un cliente sobre un tema. La calidad de servicio de una publicación que se reenvía a un suscriptor puede ser diferente a la de la publicación. Para reenviar una publicación se utiliza el valor más bajo de los dos.

Como máximo una vez
`QoS=0`

El mensaje se entrega como máximo una vez, si no, no se entrega. No se efectúa acuse de recibo la entrega del mensaje por la red.

El mensaje no se almacena. El mensaje puede perderse si se desconecta el cliente o si falla el servidor.

`QoS=0` es la modalidad de transferencia más rápida. Se denomina a veces "transmitir y olvidar".

El MQTT protocol no necesita que los servidores reenvíen publicaciones con `QoS=0` a un cliente. Si el cliente está desconectado cuando el servidor recibe la publicación, es posible que se descarte la publicación, dependiendo del servidor. El servicio de telemetría (MQXR) no descarta los mensajes enviados con `QoS=0`. Se almacenan como mensajes no persistentes y sólo se rechazan si se detiene el gestor de colas.

Al menos una vez

QoS=1

QoS=1 es el valor predeterminado la modalidad de transferencia.

El mensaje siempre se entrega, como mínimo, una vez. Si el emisor no recibe un acuse de recibo, el mensaje se envía de nuevo con el distintivo DUP establecido hasta que se reciba un acuse de recibo. Como resultado, un mismo mensaje se puede enviar varias veces al receptor y ser procesado varias veces.

El mensaje debe almacenarse localmente en el emisor y el receptor hasta que se procese.

El mensaje se suprime del receptor después de que se haya procesado. Si el receptor es un intermediario, el mensaje se publica a sus suscriptores. Si el receptor es un cliente, el mensaje se entrega a la aplicación de suscriptor. Una vez suprimido el mensaje, el receptor envía un acuse de recibo al emisor.

El mensaje se suprime del emisor después de que se haya recibido el acuse de recibo por parte del receptor.

Exactamente una vez

QoS=2

El mensaje se entrega siempre exactamente una vez.

El mensaje debe almacenarse localmente en el emisor y el receptor hasta que se procese.

QoS=2 es la modalidad de transferencia más segura, pero la más lenta. Deben realizarse como mínimo dos pares de transmisiones entre el emisor y el receptor antes de que el mensaje pueda suprimirse de la parte del emisor. El mensaje puede procesarse en el receptor tras la primera transmisión.

En el primer par de transmisiones, el emisor transmite el mensaje y obtiene acuse de recibo del receptor que ha almacenado el mensaje. Si el emisor no recibe un acuse de recibo, el mensaje se envía de nuevo con el distintivo DUP establecido hasta que se reciba un acuse de recibo.

En el segundo par de transmisiones, el emisor comunica al receptor que puede completar el proceso del mensaje, `"PUBREL"`. Si el emisor no recibe un acuse de recibo del mensaje `"PUBREL"`, el mensaje `"PUBREL"` se envía de nuevo hasta que se recibe un acuse de recibo. El emisor suprime el mensaje que ha guardado al recibir el acuse de recibo para el mensaje `"PUBREL"`

El receptor puede procesar el mensaje proporcionado en la primera o segunda fase, no tiene que volver a procesarlo. Si el receptor es un intermediario, publica el mensaje a los suscriptores. Si el receptor es un cliente, el mensaje se entrega a la aplicación de suscriptor. El receptor devuelve un mensaje de finalización al emisor para comunicarle que ha terminado de procesar el mensaje.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT

El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones

El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Publicaciones retenidas y clientes MQTT

Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Utilice el método `MqttMessage.setRetained` para especificar si una publicación en un tema está retenida.

Cuando cree o actualice una publicación retenida, envíe la publicación con una QoS de 1 o 2. Si lo envía con una QoS de 0, IBM MQ crea una publicación retenida no persistente. La publicación no se retiene si se detiene el gestor de colas.

Si publica una publicación no retenida en un tema que tiene una publicación retenida, la publicación retenida no se ve afectada. Los suscriptores actuales reciben la nueva publicación. Los suscriptores nuevos reciben la publicación retenida primero y luego las nuevas.

Puede utilizar una publicación retenida para registrar el último valor de una medida. Los nuevos suscriptores a un tema reciben inmediatamente el valor más reciente de la medida. Si no se toman nuevas medidas desde que el suscriptor se suscribió por última vez al tema de publicación, y si el suscriptor vuelve a suscribirse, el suscriptor recibe de nuevo la publicación retenida más reciente sobre el tema.

Para suprimir una publicación retenida, tiene dos opciones:

- Ejecute el mandato MQSC de **`CLEAR TOPICSTR`** .
- Crear una publicación retenida de longitud cero. Como se especifica en la especificación MQTT 3.1.1, si se publica un mensaje retenido de longitud cero en un tema, se borra cualquier mensaje retenido para dicho tema.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Suscripciones

Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos,

para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Cree suscripciones utilizando los métodos `MqttClient.subscribe`, pasando uno o varios filtros de temas y parámetros de calidades de servicio. El parámetro de calidad de servicio establece la calidad de servicio máxima que el suscriptor puede utilizar para recibir un mensaje. Los mensajes enviados a este cliente no pueden entregarse con una calidad de servicio mayor. La calidad de servicio se establece en el valor original cuando el mensaje se publicó o en el nivel especificado para la suscripción, el valor que sea más bajo de los dos. La calidad de servicio predeterminada para recibir mensajes es `QoS=1`, al menos una vez.

La propia solicitud de suscripción se envía con `QoS=1`.

Un suscriptor recibe las publicaciones cuando el cliente MQTT llama al método `MqttCallback.messageArrived`. El método `messageArrived` también pasa la serie del tema con la que el mensaje se ha publicado al suscriptor.

Puede eliminar una suscripción o conjunto de definiciones utilizando los métodos `MqttClient.unsubscribe`.

Un mandato de IBM MQ puede eliminar una suscripción.Listar suscripciones utilizando IBM MQ Explorer, o utilizando mandatos **runmqsc** o PCF. A todas las suscripciones de cliente MQTT se les asigna un nombre. Se les asigna un nombre con el formato: *`ClientIdentifier:Topic name`*

Si utiliza el `MqttConnectOptions`predeterminado, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar el cliente, las suscripciones antiguas para el cliente se eliminan cuando se conecta el cliente. Las suscripciones nuevas que el cliente realice durante la sesión se eliminan cuando se desconecta.

Si establece `MqttConnectOptions.cleanSession` en `false` antes de conectarse, las suscripciones que crea el cliente se añaden a todas las suscripciones que existían para el cliente antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otro modo de entender la forma en la que el atributo `cleanSession` afecta a las suscripciones es pensar en él como un atributo modal. Con la modalidad predeterminada, `cleanSession=true`, el cliente crea suscripciones y recibe publicaciones sólo dentro del ámbito de la sesión. En la modalidad alternativa, `cleanSession=false`, las suscripciones son duraderas. El cliente puede conectarse y desconectarse y las suscripciones permanecen activas. Cuando el cliente vuelve a conectarse, recibe las publicaciones que no se hayan entregado. Mientras está conectado, puede modificar el conjunto de suscripciones que estén activas en su nombre.

Debe establecer la modalidad `cleanSession` antes de conectarse; la modalidad dura toda la sesión. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de uso de `cleanSession=false` a `cleanSession=true`, se descartarán todas las suscripciones anteriores para el cliente y las publicaciones que no se hayan recibido.

Las publicaciones que coinciden con las suscripciones activas se envían al cliente tan pronto como se publiquen. Si el cliente está desconectado, se envían al cliente si se vuelve a conectar al mismo servidor con el mismo identificador de cliente y `MqttConnectOptions.cleanSession` establecido en `false`.

Las suscripciones de un cliente determinado se identifican por el identificador de cliente. Puede volver a conectar el cliente desde un dispositivo del cliente diferente al mismo servidor y continuar con las mismas suscripciones y recibir publicaciones que no se hayan entregado.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones

El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Series de tema y filtros de tema en clientes MQTT
Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Series de tema y filtros de tema en clientes MQTT

Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM MQ.

Las series de tema se utilizan para enviar publicaciones a suscriptores. Para crear una serie de tema, utilice el método `MqttClient.getTopic(java.lang.String topicString)`.

Los filtros de tema se utilizan para suscribirse a temas y recibir publicaciones. Los filtros de tema pueden contener comodines. Los comodines le permiten suscribirse a varios temas. Cree un filtro de tema utilizando un método de suscripción; por ejemplo, `MqttClient.subscribe(java.lang.String topicFilter)`.

Series de tema

La sintaxis de una serie de tema de IBM MQ se describe en Series de tema. La sintaxis de las series de tema de MQTT se describe en la clase `MqttClient` en la documentación de la API para el MQTT client for Java.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación del cliente MQTT.

La sintaxis de cada tipo de serie de tema es casi la misma. Existen cuatro diferencias menores:

1. Las series de tema enviadas a IBM MQ por clientes de MQTT deben seguir el convenio para los nombres de gestores de colas.
2. Las longitudes máximas difieren. Las series de tema de IBM MQ están limitadas a 10.240 caracteres. Un cliente MQTT puede crear series de tema de hasta 65535 bytes.
3. Una serie de tema creada por un cliente MQTT no puede contener ningún carácter nulo.
4. En IBM Integration Bus, un nivel de tema nulo, `'...//...'` no es válido. Los niveles de tema nulos están permitidos en IBM MQ.

A diferencia de la publicación/suscripción en IBM MQ, en el protocolo `mqttv3` no existe el concepto de objeto de tema administrativo. No puede construir una serie de tema a partir de un objeto de tema y una serie de tema. En cambio, una serie de tema está correlacionada con un tema administrativo en IBM MQ. El control de accesos asociado al tema administrativo determina si una publicación se publica en el tema o se rechaza. Los atributos que se aplican a una publicación cuando se reenvía a los suscriptores están afectados por los atributos del tema administrativo.

Filtros de tema

La sintaxis de un filtro de temas de IBM MQ se describe en Esquema de comodín basado en temas. La sintaxis de los filtros de tema que puede construir con un cliente de MQTT se describe en la clase `MqttClient` en la documentación de la API del MQTT client for Java.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación del cliente MQTT.

Conceptos relacionados

Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT
El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. En la medida de lo posible, las hebras desligan a una aplicación cliente de MQTT respecto de los retardos en los intercambios de mensajes con el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Limpiar sesiones
El cliente de MQTT y el servicio de telemetría (MQXR) mantienen la información de estado de sesión. La información de estado se utiliza para garantizar "al menos una vez" y "exactamente una vez" la entrega y "exactamente una vez" la recepción de publicaciones. El estado de la sesión también incluye las suscripciones creadas por un cliente de MQTT. Puede elegir ejecutar un cliente de MQTT con o sin el mantenimiento de información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente
El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . Cada uno de los identificadores debe ser exclusivo para un sólo cliente conectado al mismo tiempo. El identificador debe contener sólo caracteres válidos en un nombre de gestor de colas. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante tener un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con su identificador seleccionado.

Señales de entrega

Publicación de última voluntad y testamento
Si una conexión de cliente de MQTT finaliza de forma inesperada, puede configurar MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Persistencia de mensajes en clientes MQTT
Los mensajes de publicación se hacen persistentes si se envían con una calidad de servicio de "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para las publicaciones enviadas a o desde el cliente.

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. Los clientes MQTT pueden crear publicaciones para enviar a IBM MQ y suscribirse a temas en IBM MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente de MQTT proporciona tres calidades de servicio para entregar publicaciones a IBM MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Un tema puede tener una, y sólo una, publicación retenida. Si crea una suscripción a un tema que tiene una publicación retenida, la publicación se le reenvía inmediatamente.

Suscripciones
Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Desarrollo de aplicaciones Microsoft Windows Communication Foundation con IBM MQ

El canal personalizado de Microsoft Windows Communication Foundation (WCF) para IBM MQ envía y recibe mensajes entre clientes y servicios WCF.

Conceptos relacionados

“Introducción al canal personalizado de IBM MQ para WCF con .NET” en la página 1297
El canal personalizado para IBM MQ es un canal de transporte que utiliza el modelo de programación unificado de Microsoft Windows Communication Foundation (WCF).

“Utilización de canales personalizados IBM MQ para WCF” en la página 1302
Descripción general de la información disponible para los programadores que utilizan canales personalizados IBM MQ para Windows Communication Foundation (WCF).

“Utilización de los ejemplos de WCF” en la página 1322
Los ejemplos de Windows Communication Foundation (WCF) son ejemplos sencillos de cómo se puede usar un canal personalizado de IBM MQ.

FFST: Tecnología de soporte de primer error de WCF XMS

Tareas relacionadas

Rastreo del canal personalizado WCF para IBM MQ

Resolución de problemas de canal personalizado WCF para IBM MQ

Introducción al canal personalizado de IBM MQ para WCF con .NET

El canal personalizado para IBM MQ es un canal de transporte que utiliza el modelo de programación unificado de Microsoft Windows Communication Foundation (WCF).

El marco Microsoft Windows Communication Foundation framework, introducido en Microsoft.NET 3, permite desarrollar aplicaciones y servicios .NET de forma independiente del transporte y los protocolos usados para conectar con ellos, lo que permite emplear transportes o configuraciones alternativos conforme al entorno en que estén desplegados dichos servicios o aplicaciones.

WCF gestiona las conexiones en tiempo de ejecución creando una pila de canales que contiene la combinación necesaria de:

- Elementos de protocolo: Conjunto opcional de elementos de los cuales se pueden añadir ninguno, uno o más para soportar protocolos como, por ejemplo, los estándares WS-*.
- Codificador de mensajes: Elemento obligatorio en la pila que controla la serialización del mensaje en su formato de cable.
- Canal de transporte: Elemento obligatorio de la pila responsable de transportar el mensaje serializado a su punto final.

El canal personalizado para IBM MQ es un canal de transporte y, como tal, se debe emparejar con un codificador de mensajes y protocolos opcionales, según lo requiera la aplicación que utiliza un enlace personalizado de WCF. De esta forma, las aplicaciones que se han desarrollado para utilizar WCF pueden utilizar el canal personalizado para IBM MQ para enviar y recibir datos de la misma forma que utilizan los transportes incorporados proporcionados por Microsoft, lo que permite una integración sencilla con las funciones de mensajería asíncrona, escalables y fiables de IBM MQ. Para obtener una lista completa de las funciones soportadas, consulte: "Funcionalidades y prestaciones del canal personalizado WCF" en la página 1302.

¿Cuándo y por qué utilizo el canal personalizado IBM MQ para WCF?

Puede utilizar el canal personalizado IBM MQ para enviar y recibir mensajes entre clientes y servicios WCF, de la misma forma que los transportes incorporados proporcionados por Microsoft, lo que permite que las aplicaciones accedan a las características de IBM MQ dentro del modelo de programación unificado WCF.

Un patrón de uso típico para el canal personalizado de IBM MQ para WCF es como una interfaz no SOAP para la transmisión de mensajes nativos de IBM MQ .

Mensajes transportados utilizando el formato de mensaje no SOAP/no JMS (MQMessage puro)

Cuando se utiliza el canal personalizado IBM MQ para WCF como interfaz no SOAP para la transmisión de mensajes IBM MQ nativos, los mensajes se transportan utilizando el formato de mensaje no SOAP/no JMS (MQMessage puro) de IBM MQ.

Los usuarios WCF pueden iniciar el servicio, o en otras palabras, los usuarios del servicio pueden enviar un mensaje a una cola IBM MQ utilizando MQMessages. Las aplicaciones pueden obtener y establecer los campos del MQMD y la carga útil. Cuando el mensaje está disponible en colas IBM MQ , este mensaje puede ser procesado por cualquier servicio WCF o aplicaciones no WCF como, por ejemplo, aplicaciones C o Java que se ejecutan en AIX, Linux, Windowso z/OS.

Requisitos de software para el canal personalizado de IBM MQ para WCF

En este tema se describe los requisitos de software para el canal personalizado de IBM MQ para WCF. El canal personalizado IBM MQ para WCF solo puede conectarse con un gestor de colas IBM WebSphere MQ 7.0 o superior.

Requisitos del entorno de ejecución

- Microsoft.NET Framework v4.7.2 o superior debe estar instalado en la máquina host.
- *Java y .NET Messaging y servicios web* están instalados de forma predeterminada como parte del instalador de IBM MQ. Este componente instala los ensamblajes de .NET necesarios para el canal personalizado en la memoria caché de ensamblaje global.

Nota: Si Microsoft .NET Framework V4.7.2 o superior no está instalado antes de instalar IBM MQ, la instalación del producto IBM MQ continúa sin errores, pero IBM MQ classes for .NET no está disponible. Si .NET Framework se instala después de instalar IBM MQ, los ensamblajes de IBM MQ.NET se deben registrar ejecutando el script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd`, donde *WMQInstallDir*

es el directorio donde está instalado IBM MQ. Este script instala los ensamblajes necesarios en la memoria caché de ensamblaje global (GAC). Un conjunto de archivos `amqi*.log` que registran las acciones que se realizan se crean en el directorio %TEMP% . No es necesario volver a ejecutar el script `amqiRegisterdotNet.cmd` si .NET se actualiza a V4.7.2 o superior desde una versión anterior, por ejemplo, desde .NET V3.5.

Requisitos del entorno de desarrollo

- Microsoft Visual Studio 2015 o Windows Software Development Kit para .NET 4.7.2 o posterior.
- Microsoft.NET Framework V4.7.2 o superior debe estar instalado en la máquina host para poder crear los archivos de solución de ejemplo.

Canal personalizado de IBM MQ para WCF: ¿Qué está instalado?

El canal personalizado para IBM MQ es un canal de transporte que utiliza el modelo de programación unificado de Microsoft Windows Communication Foundation (WCF). De forma predeterminada, el canal personalizado se instala como parte del proceso de instalación.

Canal personalizado de IBM MQ personalizado para WCF

El canal personalizado y sus dependencias están contenidos en el componente `Java and .NET Messaging and Web Services` , que se instala de forma predeterminada. Al actualizar IBM MQ desde una versión anterior aIBM MQ 8.0, la actualización instala el canal personalizado de IBM MQ para WCF de forma predeterminada si el componente `Java and .NET Messaging and Web Services` se ha instalado previamente en una instalación anterior.

El componente `.NET Messaging and Web Services` contiene el archivo `IBM.XMS.WCF.dll` y el archivo `IBM.WMQ.WCF.dll` , y estos archivos son el conjunto de canal personalizado principal, que contiene las clases de interfaz WCF. Estos archivos se instalan en GAC (Global Assembly Cache) y también están disponibles en el directorio siguiente: *MQ_INSTALLATION_PATH* `\bin`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

La siguiente tabla resume las clases clave necesarias para utilizar el canal personalizado.

Tabla 189. Las clases clave necesarias para utilizar el canal personalizado

	Interfaz SOAP/JMS	**Interfaz No-SOAP/No-JMS (A partir de IBM MQ 8.0)**
Ensamblaje de canal personalizado	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Nombre de enlace de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Importador de enlaces de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Config. enlaces de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Ejemplos (OneWay)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client,MQMessaging_OneWay_Service
Ejemplos (RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client,MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll da soporte a las interfaces SOAP/JMS y No-SOAP/No-JMS. Se recomienda que las nuevas aplicaciones desarrolladas utilicen el ensamblaje IBM.WMQ.WCF, ya que soporta ambas interfaces.

Envío de mensajes MQSTR con formato

Si el mensaje de solicitud es de tipo MQSTR, puede seleccionar el formato MQSTR para enviar el mensaje.

Debe utilizar un parámetro de URI adicional, **replyMessageFormat**, para cambiar el formato del mensaje de respuesta. Los valores soportados son:

""

" " es el valor predeterminado.

El mensaje de respuesta está en formato de byte (MQMFT_NONE). Por ejemplo:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

El mensaje de respuesta está en formato MQSTR (MQMFT_STRING). Por ejemplo:

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

Notas:

1. El valor de **replyMessageFormat** distingue entre mayúsculas y minúsculas.
2. Si se utiliza un valor distinto a " " o *MQSTR*, se genera una excepción de valor de parámetro no válido.

Ejemplos de canal personalizado de IBM MQ

Los ejemplos proporcionan algunos ejemplos sencillos sobre cómo se puede utilizar el canal personalizado de IBM MQ para WCF. Los ejemplos y sus archivos asociados se encuentran en el directorio *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ. Para obtener más información sobre los ejemplos del canal personalizado de IBM MQ, consulte "Utilización de los ejemplos de WCF" en la página 1322.

svcutil.exe.config

`svcutil.exe.config` es un ejemplo de los valores de configuración necesarios para habilitar la herramienta de generación de proxy de cliente Microsoft WCF `svcutil` para reconocer el canal personalizado. El archivo `svcutil.exe.config` se encuentra en el directorio *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ. Para obtener más información sobre cómo utilizar el archivo `svcutil.exe.config`, consulte la sección "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con metadatos de un servicio en ejecución" en la página 1319.

Arquitectura WCF

El canal personalizado de IBM MQ para WCF se integra encima de la API `IBM Message Service Client for .NET (XMS .NET)`.

Interfaz SOAP/JMS

La arquitectura WCF es la que se muestra en el diagrama siguiente:

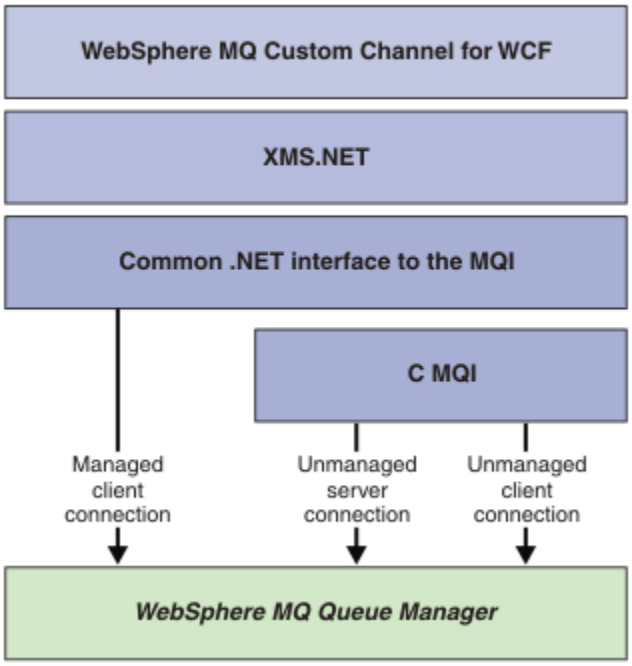


Figura 149. Arquitectura WCF para la interfaz SOAP/JMS

De forma predeterminada, todos los componentes necesarios se instalan con el producto.

Las tres conexiones son:

- Conexiones de cliente gestionado
- Conexiones de servidor no gestionadas.
- Conexiones de cliente no gestionadas.

Para obtener más información sobre estas conexiones, consulte “Opciones de conexión WCF” en la página 1308.

Interfaz No-SOAP/No-JMS

El canal personalizado de IBM MQ para WCF da soporte a la interfaz SOAP/JMS (disponible desde IBM WebSphere MQ 7.0.1) y, también, a la interfaz no SOAP/no JMS.

La arquitectura WCF es la que se muestra en el diagrama siguiente:

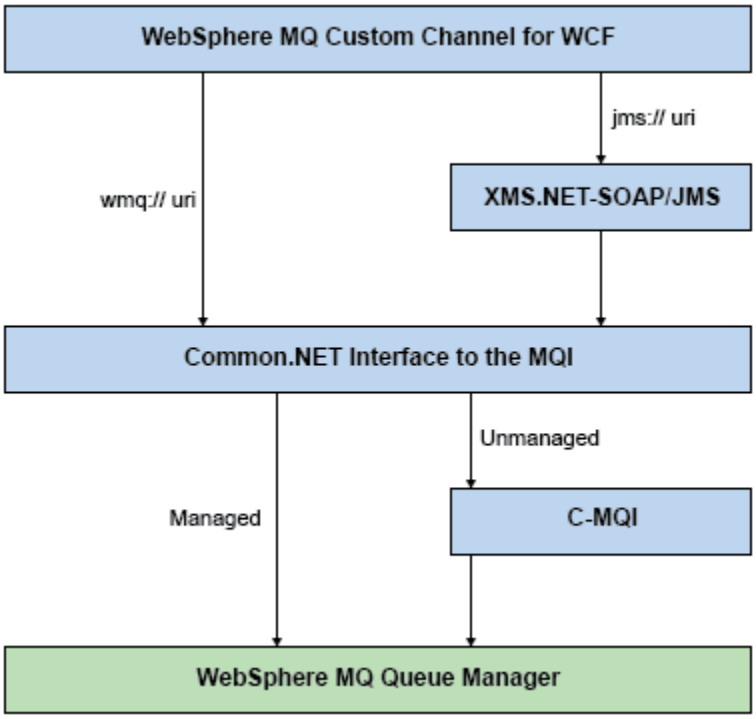


Figura 150. Arquitectura WCF para la interfaz no SOAP/no JMS

Utilización de canales personalizados IBM MQ para WCF

Descripción general de la información disponible para los programadores que utilizan canales personalizados IBM MQ para Windows Communication Foundation (WCF).

Microsoft Windows Communication Foundation apoya los servicios web y el soporte de mensajería en Microsoft.NET Framework 3. IBM MQ se puede utilizar como un canal personalizado dentro de WCF en .NET Framework 3 de la misma forma que los canales incorporados que ofrece Microsoft.

Los mensajes transportados a través del canal personalizado se formatean de acuerdo con la implementación de SOAP sobre JMS de IBM MQ. A continuación, las aplicaciones pueden comunicarse con los servicios alojados por WCF o por la infraestructura de servicio WebSphere SOAP sobre JMS .

Funcionalidades y prestaciones del canal personalizado WCF

Utilice los temas siguientes para obtener información sobre las funciones y prestaciones del canal personalizado WCF.

Formas de canal personalizado de WCF

Descripción general de las formas de canal personalizado que IBM MQ puede utilizar como dentro de los canales personalizados de Microsoft Windows Communication Foundation (WCF).

El canal personalizado de IBM MQ para WCF admite dos formas de canal:

- unidireccional
- solicitud-respuesta

WCF selecciona automáticamente la forma de canal de acuerdo con el contrato de servicio que se esté alojando.

Los contratos que incluyen métodos que sólo utilizan el parámetro **IsOneWay** son atendidos por la forma de un canal unidireccional, por ejemplo:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Los contratos que incluyen una combinación de métodos unidireccional y de solicitud-respuesta, o todos los métodos de solicitud-respuesta, son atendidos por la forma de canal de solicitud-respuesta. Por ejemplo:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Nota: Al mezclar métodos unidireccionales y de solicitud/respuesta en el mismo contrato, debe asegurarse de que el comportamiento el esperado, especialmente cuando trabaje en un entorno mixto, porque los métodos unidireccionales esperan hasta recibir una respuesta nula desde el servicio.

Canal unidireccional

El canal personalizado unidireccional de IBM MQ para WCF se utiliza, por ejemplo, para enviar mensajes desde un cliente WCF utilizando una forma de canal unidireccional. El canal puede enviar mensajes en una sola dirección, por ejemplo, de un gestor de colas de cliente a una cola en un servicio WCF.

Canal de solicitud/respuesta

El canal personalizado de solicitud/respuesta de IBM MQ para WCF se utiliza, por ejemplo, para enviar mensajes en dos direcciones de forma asíncrona; la misma instancia de cliente se debe utilizar para la mensajería asíncrona. El canal puede enviar mensajes en una dirección, por ejemplo; de un gestor de colas de cliente a una cola de un servicio WCF y, a continuación, enviar un mensaje de respuesta de WCF a una cola en el gestor de colas del cliente.

Nombres y valores de los parámetros de un URI de WCF

Nombres y valores de parámetros de URI de las interfaces SOAP/JMS y no-SOAP/no-JMS.

Interfaz SOAP/JMS

connectionFactory
El parámetro connectionFactory es obligatorio.

initialContextFactory
El parámetro initialContextFactory es obligatorio y hay que establecerlo en "com.ibm.mq.jms.Nojndi" para que sea compatible con WebSphere Application Server y otros productos.

Interfaz no-SOAP/no-JMS

El formato de URI es el descrito en las especificaciones MA93. Consulte SupportPac - MA93 para obtener detalles de las especificaciones IRI de IBM MQ.

Sintaxis de URI de IBM MQ

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
```

```
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

Ejemplo de IRI de IBM MQ

El siguiente ejemplo de IRI indica a un solicitante de servicio que puede utilizar una conexión de enlace de cliente TCP de IBM MQ con una máquina llamada example.com por el puerto 1414 y colocar mensajes de petición persistentes en una cola llamada SampleQ del gestor de colas QM1. El IRI especifica que el proveedor de servicios colocará las respuestas a una cola llamada SampleReplyQ.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

En conexiones habilitadas para TLS

Para establecer conexiones seguras (TLS) utilizando el cliente/servicio WCF, establezca las propiedades siguientes con los valores apropiados en el URI. Todas las propiedades prefijadas con "*" son obligatorias para establecer una conexión segura.

- **sslKeyRepository**: *SYSTEM o *USER
- * **sslCipherSpec**: una CipherSpec válida, por ejemplo TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: true o false.
- **sslKeyResetCount**: un valor mayor que 32kb.
- **sslPeerName:** el nombre distinguido del certificado de servidor.

Por ejemplo:

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

Entrega garantizada del canal personalizado WCF

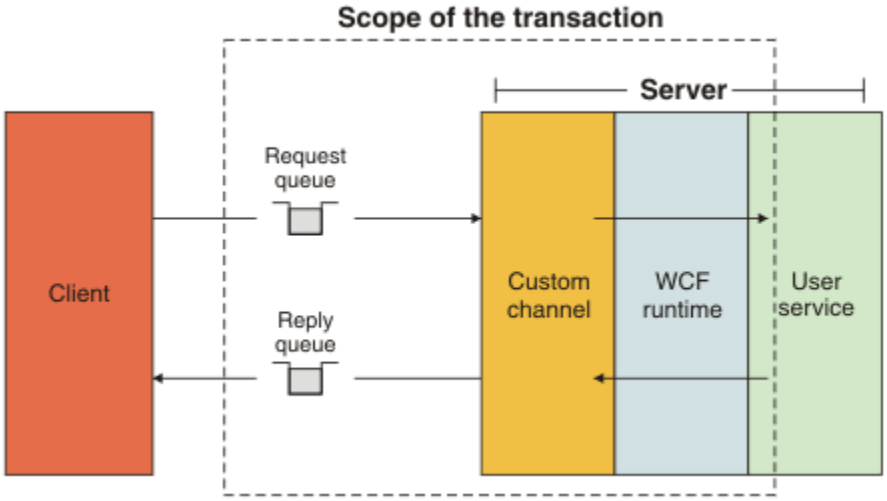
Una entrega garantizada garantiza que una solicitud o respuesta de servicio se accione y no se pierda.

Se recibe un mensaje de solicitud y cualquier mensaje de respuesta se envía bajo un punto de sincronización de transacción local, que se puede retrotraer en el caso de error en tiempo de ejecución. Son ejemplos de tales errores: una excepción no manejada lanzada por un servicio o no entregar el mensaje al servicio o el mensaje de respuesta.

`AssuredDelivery` es el atributo de entrega garantizada que se puede especificar en un contrato de servicio para garantizar que cualquier mensaje de solicitud recibido por un servicio y cualquier mensaje de respuesta enviado desde un servicio no se pierdan en caso de un error en tiempo de ejecución.

Para asegurarse de que los mensajes también se conservan en caso de error del sistema o caída de tensión, los mensajes han de enviarse como persistentes. Para utilizar mensajes permanentes, la aplicación cliente ha de tener esta opción especificada en el URI de punto final.

Las transacciones distribuidas no están soportadas, y el ámbito de la transacción no se extiende más allá del proceso de mensajes de solicitud y respuesta realizado por IBM MQ. Cualquier trabajo realizado dentro del servicio se podría volver a ejecutar como consecuencia de un fallo, lo que haría que el mensaje se recibiera de nuevo. El diagrama siguiente muestra el ámbito de la transacción:



La entrega garantizada se habilita aplicando el atributo `AssuredDelivery` a la clase de servicio, tal como se muestra en el ejemplo siguiente:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Cuando se utiliza el atributo `AssuredDelivery`, hay que tener en cuenta los puntos siguientes:

- Cuando un canal determina que es probable que un fallo se repita si cuando un mensaje se retrotrae y se vuelve a recibir, dicho el mensaje se trata como un mensaje envenenado y no se devuelve a la cola de solicitudes para su reprocesamiento. Por ejemplo: si el mensaje recibido no tiene el formato correcto o no se puede asignar a un servicio. Las excepciones no manejadsa lanzadas desde una operación de servicio siempre se reenvían hasta alcanzar el número máximo de veces especificado en la propiedad de umbral de restitución de la cola de solicitudes. Para obtener más información, consulte: “Mensajes con formato incorrecto del canal personalizado WCF” en la página 1306
- El canal realiza la lectura, el procesamiento y la respuesta de cada mensaje de solicitud como una operación atómica utilizando una sola hebra de ejecución para forzar la integridad transaccional. Para permitir que las operaciones de servicio ejecuten de de forma concurrente, el canal habilita WCF para crear varias instancias del canal. El número de instancias de canal disponibles a las solicitudes de proceso se controla mediante la propiedad de enlace `MaxConcurrentCalls`. Para obtener más información, consulte: “Opciones de configuración de enlace WCF” en la página 1314
- La función de entrega garantizada utiliza los puntos de extensibilidad WCF IOperationInvoker e IErrorHandler. Si una aplicación utiliza externamente estos puntos de extensibilidad, habrá de asegurarse de que se invoquen los puntos de extensibilidad que se hayan registrado anteriormente. Si no se hace, IErrorHandler puede dar como resultado errores que no se notifican. Si no se hace, IOperationInvoker puede hacer que WCF deje de responder.

Seguridad de un canal personalizado WCF

El canal personalizado de IBM MQ para WCF admite el uso de TLS solo para las conexiones cliente no gestionadas con el gestor de colas.

Usando una entrada en la tabla de definiciones de canal de cliente (CCDT). Para obtener más información sobre las CCDT, consulte Tabla de definición de canal de cliente.

Tablas de definiciones de canal de cliente (Client Channel Definition Table, CCDT) de WCF

El canal personalizado IBM MQ para WCF da soporte al uso de las tablas de definiciones de canal de cliente (CCDT) para configurar la información de conexión para las conexiones de cliente.

Las CCDT se controlan mediante estas dos variables de entorno:

- *MQCHLLIB* especifica el directorio en el que se encuentra la tabla.
- *MQCHLTAB* especifica el nombre de archivo de la tabla.

Si se definen estas variables de entorno, entonces tendrán prioridad sobre cualquier detalle de conexión cliente especificado en el URI.

Para obtener más información sobre las tablas de definiciones de canal de cliente, consulte Tabla de definiciones de canal de cliente.

Mensajes con formato incorrecto del canal personalizado WCF

Cuando un servicio no puede procesar un mensaje de solicitud, o no puede entregar un mensaje de respuesta a una cola de respuestas, el mensaje se trata como un mensaje con formato incorrecto.

Mensajes de solicitud con formato incorrecto

Si no se puede procesar un mensaje de solicitud, se trata como un mensaje con formato incorrecto. Esta acción impide que el servicio reciba de nuevo el mismo mensaje no procesable. Para que un mensaje de solicitud no procesable pueda tratarse como un mensaje con formato incorrecto, debe cumplirse una de las siguientes soluciones:

- El recuento de restituciones de mensajes ha superado el umbral de restitución especificado en la cola de solicitudes, lo que solo ocurre si se ha especificado una entrega garantizada para el servicio. Para obtener más información sobre la entrega garantizada, consulte: “Entrega garantizada del canal personalizado WCF” en la página 1304
- El mensaje no se ha formateado correctamente y no se ha podido interpretar como un mensaje SOAP sobre JMS.

Mensajes de respuesta con formato incorrecto

Si un servicio no puede entregar un mensaje de respuesta a la cola de respuestas, el mensaje de respuesta se trata como un mensaje con formato incorrecto. Para los mensajes de respuesta, esta acción permite recuperar posteriormente los mensajes de respuesta para ayudar a la determinación de problemas.

Manejo de mensajes con formato incorrecto

La acción realizada para un mensaje con formato incorrecto depende de la configuración del gestor de colas y de los valores establecidos en las opciones de informe del mensaje. Para SOAP sobre JMS, se establecen las siguientes opciones de informe en los mensajes de solicitud de forma predeterminada, que no son configurables:

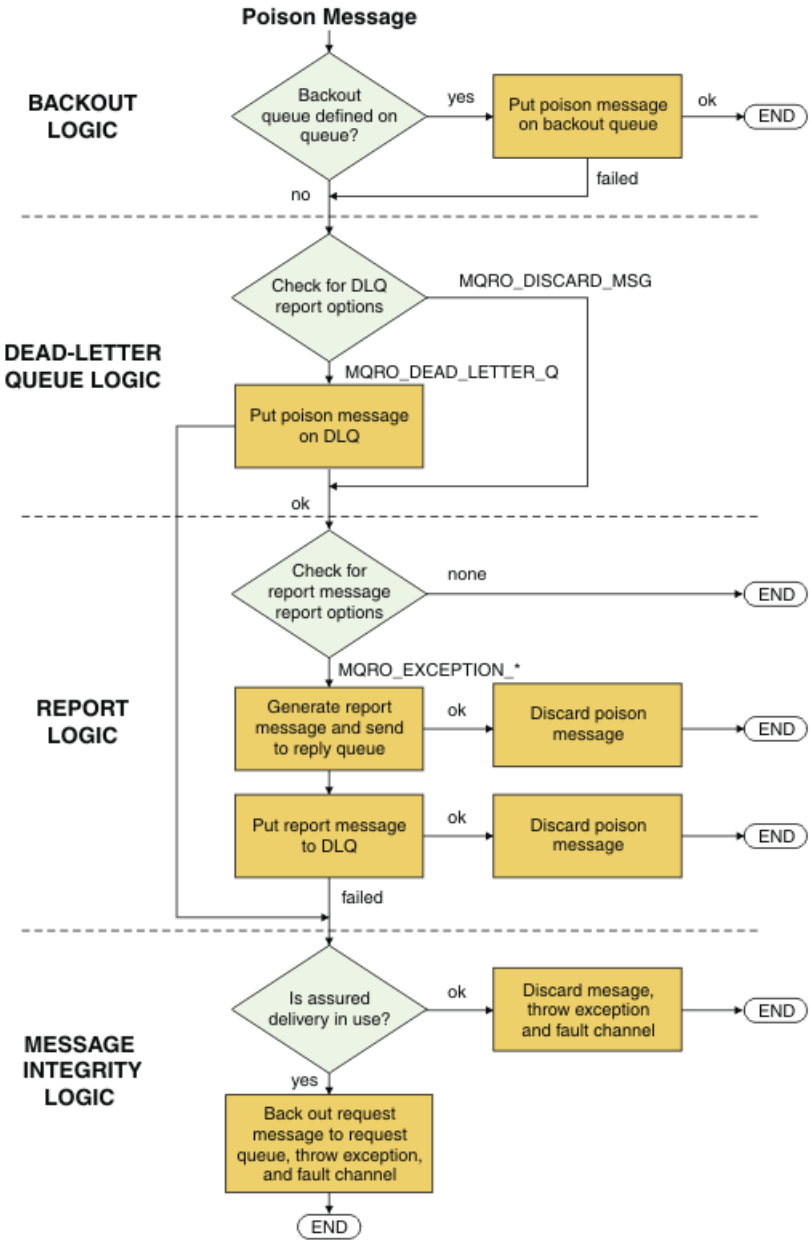
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Para SOAP sobre JMS, se establece la siguiente opción de informe en los mensajes de respuesta de forma predeterminada, que no es configurables:

- MQRO_DEAD_LETTER_Q

Si los mensajes provienen de un origen no WCF, consulte la documentación del origen.

El diagrama siguiente muestra las acciones posibles y los pasos realizados si falla el manejo de mensajes con formato incorrecto:



Prestaciones de mensajes de IBM MQ para aplicaciones WCF

Prestaciones de mensajes no SOAP/no JMS (es decir, IBM MQ) para aplicaciones WCF.

Para la interfaz no SOAP/no JMS, las prestaciones de mensajes IBM MQ para aplicaciones WCF son las siguientes:

- Las aplicaciones WCF pueden enviar y recibir los mensajes IBM MQ base que pueden ser procesados por cualquier aplicación IBM MQ.
- Las aplicaciones WCF tienen completo control para actualizar el MQMD y la carga útil.
- El cliente WCF puede enviar mensajes IBM MQ que pueden ser consumidos por cualquier cliente IBM MQ, por ejemplo clientes C, Java, JMS y .NET.

WCF para la interfaz no SOAP/no JMS debe utilizar las clases siguientes para establecer la carga útil del mensaje y MQMD para el mensaje:

- WmqStringMessage para una carga útil de tipo String.
- WmqBytesMessage para una carga útil de tipo Bytes.
- WmqXmlMessage para una carga útil de tipo XML.

Para configurar la carga útil del mensaje, use la propiedad **Data** de las clases WmqStringMessage, WmqBytesMessage o WmqXmlMessage, dependiendo del tipo de carga útil. Por ejemplo, utilice el código siguiente para establecer una carga útil de tipo String:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

Opciones de conexión WCF

Hay tres modalidades de conexión de un canal personalizado de IBM MQ para WCF a un gestor de colas. Estudie qué tipo de conexión se ajusta mejor a sus requisitos.

Para obtener más información sobre las opciones de conexión, consulte: “Diferencias de conexión” en la página 596

Para obtener más información sobre la arquitectura WCF, consulte: “Arquitectura WCF” en la página 1300

Conexión de cliente no gestionado

Una conexión realizada en este modo se conecta como un cliente de IBM MQ a un servidor de IBM MQ que se ejecuta en la máquina local o en una remota.

Para utilizar el canal personalizado de IBM MQ para WCF como un cliente de IBM MQ, puede instalarlo, con el IBM MQ MQI client, en el servidor de IBM MQ o en una máquina aparte.

Conexión de servidor no gestionado

Cuando se utiliza en la modalidad de enlaces de servidor, el canal personalizado de IBM MQ para WCF utiliza la API del gestor de colas, en lugar de comunicarse a través de una red. El uso de conexiones de enlaces proporciona un mejor rendimiento para las aplicaciones de IBM MQ que el uso de conexiones de red.

Para utilizar la conexión de enlaces, debe instalar el canal personalizado de IBM MQ para WCF en el servidor de IBM MQ.

Conexión de cliente gestionado

Una conexión realizada en este modo se conecta como un cliente de IBM MQ a un servidor de IBM MQ que se ejecuta en la máquina local o en una remota.

Las clases de canal personalizado de IBM MQ para .NET que se conecta en esta modalidad permanece en el código gestionado de .NET y no hace llamadas a los servicios nativos. Para obtener más información sobre código gestionado, consulte la documentación de Microsoft.

Hay una serie de limitaciones para utilizar el cliente gestionado. Para obtener más información sobre dichas limitaciones, consulte “Conexiones de cliente gestionado” en la página 596.

Creación y configuración del canal personalizado IBM MQ para WCF

Los canales personalizados IBM MQ para WCF funcionan de la misma forma que los canales WCF de transporte que ofrece Microsoft. El canal personalizado IBM MQ para WCF se puede crear de una de dos formas.

Acerca de esta tarea

El canal personalizado IBM MQ integra WCF como un canal de transporte WCF y, como tal, se debe emparejar con un codificador de mensajes y canales de protocolo adicionales, de modo que pueda crear una pila de canales completa que pueda utilizar una aplicación. Se necesitan dos elementos para que una pila de canal completa se cree correctamente:

1. Una definición de enlace: especifica qué elementos se necesitan para crear la pila de canales de aplicaciones, incluido el canal de transporte, el codificador de mensajes y cualquier protocolo, además de los valores de configuración generales. En un canal personalizado, la definición de enlace se tiene que crear en forma de un enlace personalizado WCF.
2. Una definición de punto final: enlaza el contrato de servicio con la definición de enlace y también proporciona el URI de conexión real que describe dónde se puede conectar la aplicación. En un canal personalizado, el URI es de la forma SOAP sobre JMS.

Estas definiciones pueden crearse de dos maneras diferentes:

- Administrativamente: las definiciones se crean proporcionando los detalles en un archivo de configuración de aplicación (por ejemplo: `app.config`).
- Programmáticamente: las definiciones se crean directamente desde el código de la aplicación.

La decisión sobre qué método utilizar para crear las definiciones ha de basarse en los requisitos de la aplicación, tal como se indica a continuación:

- El método administrativo de configuración proporciona la flexibilidad necesaria para modificar los detalles del servicio y el posterior despliegue del cliente sin volver a compilar la aplicación.
- El método programático de configuración proporciona una mayor protección frente a errores de configuración y la capacidad de generar dinámicamente una configuración en tiempo de ejecución.

Creación de un canal personalizado WCF administrativamente suministrando la información de enlace y punto final en un archivo de configuración de aplicaciones

El canal personalizado de IBM MQ para WCF es un canal WCF de nivel de transporte. Se deben definir un punto final y un enlace para utilizar el canal personalizado, y estas definiciones pueden realizarse suministrando la información de enlace y de punto final en un archivo de configuración de aplicación.

Para configurar y utilizar el canal personalizado de IBM MQ para WCF, que es un canal WCF de nivel de transporte, deben definirse un enlace y una definición de punto final. El enlace contiene la información de configuración del canal y la definición de punto final contiene los detalles de la conexión. Estas definiciones pueden crearse de dos formas:

- Mediante programación, directamente a partir del código de aplicación, tal como se describe aquí: "Creación de un canal personalizado WCF suministrando la información de enlace y punto final mediante programación" en la página 1311
- Administrativamente, proporcionando los detalles en un archivo de configuración de aplicación, tal como se describe en el siguiente procedimiento.

El archivo de configuración de la aplicación de cliente o servicio se denomina comúnmente *yourappname*`.exe.config` donde *yourappname* es el nombre de la aplicación. El archivo de configuración de aplicación se modifica más fácilmente utilizando la herramienta del editor de configuración de servicio de Microsoft denominada `SvcConfigEditor.exe` de la siguiente forma:

- Inicie la herramienta del editor de configuración `SvcConfigEditor.exe`. La ubicación de instalación predeterminada para la herramienta es: *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` donde *Unidad:* es el nombre de la unidad de instalación.

Paso 1: Añadir una extensión de elemento de enlace para habilitar WCF para localizar el canal personalizado

1. Pulse con el botón derecho del ratón en **Avanzado** > **Extensión** > **elemento de enlace** para abrir el menú y seleccione **Nuevo**
2. Rellene los campos como se muestra en esta tabla:

Tabla 190. Campos de Nuevo elemento de enlace	
Campo	**Valor**
Nombre	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
Type	Vaya a `IBM.XMS.WCF.dll` en Global Assembly Cache (GAC) y seleccione `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

Paso 2: Crear una definición de enlace personalizado que empareja el canal personalizado con un codificador de mensajes WCF

1. Pulse con el botón derecho del ratón en **Enlaces** para abrir el menú y seleccione **Nueva configuración de enlace**
2. Rellene los campos como se muestra en esta tabla:

Tabla 191. Campos de Nueva configuración de enlace	
Campo	**Valor**
Nombre	`CustomBinding_WMQ`
BindingElement 1	`textMessageEncoding (MessageVersion: Soap11)`
BindingElement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Paso 3: Especificar las propiedades de enlace

1. Seleccione el enlace de transporte *IBM.XMS.WCF.SoapJmsIbmTransportChannel* en el enlace que ha creado en: “Paso 2: Crear una definición de enlace personalizado que empareja el canal personalizado con un codificador de mensajes WCF” en la página 1310
2. Realice los cambios necesarios en los valores predeterminados de las propiedades, tal como se describe en: “Opciones de configuración de enlace WCF” en la página 1314

Paso 4: Crear una definición de punto final

Cree una definición de punto final que haga referencia al enlace personalizado que ha creado en: “Paso 2: Crear una definición de enlace personalizado que empareja el canal personalizado con un codificador de mensajes WCF” en la página 1310 y proporcione los detalles de la conexión del servicio. La forma en la que se especifica esta información depende de si la definición es para una aplicación cliente o para una aplicación de servicio.

Para una aplicación cliente, añada una definición de punto final a la sección de cliente, tal como se indica a continuación:

1. Pulse con el botón derecho del ratón en **Cliente** > **Puntos finales** para abrir el menú y seleccione **Nuevo punto final de cliente**
2. Rellene los campos como se muestra en esta tabla:

Tabla 192. Campos de Nuevo punto final de cliente	
Campo	**Valor**
Nombre	`Endpoint_WMQ`

Tabla 192. Campos de Nuevo punto final de cliente (continuación)	
Campo	**Valor**
Dirección	*El URI de SOAP/JMS que describe los detalles de conexión WMQ necesarios para acceder al servicio.* Para obtener más detalles, consulte: "Canal personalizado de IBM MQ para el formato de dirección de URI del punto final WCF" en la página 1313
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract	*El nombre de la interfaz de contrato de servicio*

Para una aplicación de servicio, añada una definición de servicio a la sección de servicios de la siguiente manera:

1. Pulse con el botón derecho **Servicios** para abrir el menú y seleccione **Nuevo servicio**. A continuación, seleccione la clase de servicio que se va a alojar.
2. Añada una definición de punto final a la sección **Puntos finales** para el nuevo servicio y rellene los campos como se muestra en esta tabla:

Tabla 193. Campos de puntos finales de Nuevo servicio	
Campo	**Valor**
Nombre	`Endpoint_WMQ`
Dirección	*El URI de SOAP/JMS que describe los detalles de conexión WMQ necesarios para acceder al servicio.* Para obtener más detalles, consulte: "Canal personalizado de IBM MQ para el formato de dirección de URI del punto final WCF" en la página 1313
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract	*El nombre de la clase de implementación de servicio*

Creación de un canal personalizado WCF suministrando la información de enlace y punto final mediante programación

El canal personalizado de IBM MQ para WCF es un canal WCF de nivel de transporte. Se deben definir un punto final y un enlace para utilizar el canal personalizado, y estas definiciones pueden realizarse mediante programación directamente desde el código de aplicación.

Para configurar y utilizar el canal personalizado de IBM MQ para WCF, que es un canal WCF de nivel de transporte, deben definirse un enlace y una definición de punto final. El enlace contiene la información de configuración del canal y la definición de punto final contiene los detalles de la conexión. Para obtener más información, consulte el apartado "Utilización de los ejemplos de WCF" en la página 1322.

Estas definiciones pueden crearse de dos formas:

- Administrativamente, proporcionando los detalles en un archivo de configuración de aplicación, tal como se describe en "Creación de un canal personalizado WCF administrativamente suministrando la información de enlace y punto final en un archivo de configuración de aplicaciones" en la página 1309.
- Mediante programación, directamente a partir del código de aplicación, tal como se describe en los subtemas siguientes.

Definición programática de información de enlace y punto final: interfaz SOAP/JMS
Para la interfaz SOAP/JMS, puede definir mediante programa un punto final y un enlace directamente desde el código de la aplicación.

Acerca de esta tarea

Para suministrar programáticamente información de enlace y punto final, añada el código necesario a la aplicación siguiendo los pasos siguientes:

Procedimiento

1. Cree una instancia del elemento de enlace de transporte del canal añadiendo el código siguiente a la aplicación:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. Establezca las propiedades de enlace necesarias, por ejemplo, añadiendo el código siguiente a la aplicación para establecer `ClientConnectionMode`:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. Cree un enlace personalizado que empareje el canal de transporte con un codificador de mensajes añadiendo el código siguiente a la aplicación:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. Cree el URI de SOAP/JMS.

   El URI de SOAP/JMS que describe los detalles de conexión de IBM MQ necesarios para acceder al servicio, se debe proporcionar como la dirección de punto final. La dirección que se especifique dependerá de si el canal se va a usar para una aplicación de servicio o de cliente.

   - Para las aplicaciones cliente, el URI de SOAP/JMS se debe crear como un `EndpointAddress` de la forma siguiente:

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - Para las aplicaciones de servicio, el URI de SOAP/JMS se debe crear como un URI de la forma siguiente:

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     Para obtener más información sobre las direcciones de punto final, consulte “Canal personalizado de IBM MQ para el formato de dirección de URI del punto final WCF” en la página 1313.

Definición programática de la información de enlace y de punto final: interfaz no SOAP/no JMS
Para la interfaz no SOAP/no JMS, puede definir programáticamente un punto final y un enlace directamente desde el código de la aplicación.

Acerca de esta tarea

Para suministrar programáticamente información de enlace y punto final, añada el código necesario a la aplicación siguiendo los pasos siguientes:

Procedimiento

1. Cree un WmqBinding añadiendo el código siguiente a la aplicación:

   ```
   WmqBinding binding = new WmqBinding();
   ```

   Este código crea un enlace que empareja WmqMsgEncodingElement y WmqIbmTransportBindingElement necesarios para la interfaz no SOAP/no JMS.
2. Proporcione el URI wmq:// que describe los detalles de conexión IBM MQ necesarios para acceder al servicio.

   La forma en la que se proporcione el URI wmq:// dependerá de si el canal se utiliza para una aplicación de servicio o una aplicación cliente.

   - En aplicaciones cliente, hay que crear el URI wmq:// como EndpointAddress de la forma siguiente:

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - En aplicaciones de servicio, hay que crear el URI wmq:// de la forma siguiente:

     ```
     Uri sampleAddress = new Uri(
     "wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

Canal personalizado de IBM MQ para el formato de dirección de URI del punto final WCF

Un servicio web se especifica mediante un identificador de recurso universal (URI) que proporciona los detalles de ubicación y conexión. El formato de URI depende de si está utilizando la interfaz SOAP/JMS o la interfaz no SOAP/no JMS.

Interfaz SOAP/JMS

El formato de URI que está soportado en el transporte IBM MQ para SOAP permite obtener un amplio control sobre los parámetros y las opciones específicos de SOAP/ IBM MQ al acceder a los servicios de destino. Este formato es compatible con WebSphere Application Server y con CICS, lo que facilita la integración de IBM MQ con ambos productos.

La sintaxis del URI es la siguiente:

```
jms:/queue? name=value&name=value...
```

donde `nombre` es un nombre de parámetro y *valor* es un valor adecuado, y el elemento `nombre` = *valor* se puede repetir cualquier número de veces con la segunda y subsiguientes apariciones precedidas por un ampersand (&).

Los nombres de parámetro distinguen entre mayúsculas y minúsculas, ya que son nombres de objetos IBM MQ. Si se especifica un parámetro más de una vez, la aparición final del parámetro será la que entre en vigor, lo que significa que las aplicaciones cliente pueden sustituir un valor de parámetro añadiéndolo al URI. Si se incluye algún parámetro no reconocido adicional, se ignorará.

Si se almacena un URI en una cadena XML, hay que representar el carácter ampersand como "&". De forma similar, si un URI está codificado en un script, tenga cuidado de escapar caracteres como **&** que de otro modo serían interpretados por el shell.

Esto es un ejemplo de URI simple de un servicio Axis:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

A continuación, se muestra un ejemplo de un URI simple para un servicio .NET:

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Solo se proporcionan los parámetros necesarios (`targetService` es necesario solo para los servicios .NET), y a `connectionFactory` no se le proporciona ninguna opción.

En este ejemplo de Axis, `connectionFactory` contiene una serie de opciones:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

En este ejemplo de Axis, también se ha especificado la opción `sslPeerName` de `connectionFactory`. El propio valor de sslPeerName contiene pares nombre-valor e incluye espacios en blanco significativos:

```
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
```

Interfaz No-SOAP/No-JMS

El formato de URI de la interfaz No-SOAP/No-JMS ofrece un alto grado de control de los parámetros y opciones específicos de IBM MQ.

La sintaxis del URI es la siguiente:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

Este IRI indica a un solicitante de servicio que puede utilizar una conexión de enlace de cliente TCP de IBM MQ con una máquina llamada example.com por el puerto 1415 y colocar mensajes de petición persistentes en una cola llamada INS.QUOTE.REQUEST del gestor de colas MOTOR.INS. El IRI especifica que el proveedor de servicios coloca las respuestas en una cola llamada INS.QUOTE.REPLY en el gestor de colas BRANCH452. El formato de URI es el especificado para SupportPac MA93. Consulte SupportPac MA93: IBM MQ - Definición de servicio para obtener más detalles sobre las especificaciones IRI de IBM MQ.

Opciones de configuración de enlace WCF

Hay dos maneras de aplicar opciones de configuración a la información de enlace de canales personalizados. Puede establecer las propiedades de manera administrativa o establecerlas mediante programación.

Las opciones de configuración de enlace se pueden establecer de dos maneras distintas:

1. De manera administrativa: los valores de las propiedades de enlace deben especificarse en la sección de transporte de la definición de enlace personalizada en el archivo de configuración de las aplicaciones, por ejemplo: `app.config`.
2. Mediante programación: debe modificarse el código de la aplicación para especificar la propiedad durante la inicialización del enlace personalizado.

Establecimiento de las propiedades de enlace de forma administrativa

Los valores de las propiedades de enlace se pueden especificar en el archivo de configuración de la aplicación, por ejemplo: `app.config`. El archivo de configuración se genera mediante **`svcutil`**, tal como se muestra en los ejemplos siguientes.

Interfaz SOAP/JMS

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Interfaz No-SOAP/No-JMS

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Establecimiento de las propiedades de enlace mediante programación

Para añadir una propiedad de enlace WCF para especificar la modalidad de conexión del cliente, debe modificar el código de servicio para especificar la propiedad durante la inicialización del enlace personalizado.

Utilice el ejemplo siguiente para especificar la modalidad de conexión de cliente no gestionado:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

Propiedades de enlace WCF

Tabla 194. Valores de las propiedades de enlace cuando se establecen de forma administrativa o mediante programación

Nombre de propiedad	Aplicación cliente o de servicio	Valor administrativo	Valor programático	Descripción
maxBufferPoolSize	Ambos	Entero con signo de 0 a 64 bits	Entero con signo de 0 a 64 bits	Especifica el tamaño máximo de la memoria que se puede utilizar para almacenar almacenamientos intermedios de mensajes WCF para una instancia del canal.
maxMessageSize	Ambos	Entero con signo de 1 a 32 bits	Entero con signo de 1 a 32 bits	Especifica la memoria máxima que se puede utilizar para un mensaje WCF individual.

Tabla 194. Valores de las propiedades de enlace cuando se establecen de forma administrativa o mediante programación (continuación)

Nombre de propiedad	Aplicación cliente o de servicio	Valor administrativo	Valor programático	Descripción
clientConnectionMode	Ambos	0 (valor predeterminado) 1	AS_URI (valor predeterminado) CLIENT_UNMANAGED	Especifica la modalidad de conexión de cliente del canal de transporte. 0 implica que la modalidad de conexión de cliente es la que se especifica en el URI. Solo se utiliza si se usa la conexión de cliente. Especifica que la modalidad de conexión de cliente es la que se especifica en el URI. 0 es el valor predeterminado si no se establece ninguna modalidad de conexión de cliente. 1 implica que la modalidad de conexión de cliente es un cliente no gestionado. Solo se utiliza si se usa la conexión de cliente.
MaxConcurrentCalls	Cliente	El rango es 0 - 2 147 483 647 16 es el valor predeterminado	El rango es 0 - 2 147 483 647 16 es el valor predeterminado	Esta propiedad define el número máximo de operaciones simultáneas que se pueden llevar a cabo en un proxy de cliente individual en un momento determinado. Si se inician más operaciones, se pondrán en cola hasta que finalice una operación en curso o se agote el tiempo de espera. Este valor se puede utilizar para controlar las hebras y recursos máximos que puede consumir un proxy individual. 0 elimina este límite, permitiendo que todas las operaciones se intenten de forma simultánea.

Tabla 194. Valores de las propiedades de enlace cuando se establecen de forma administrativa o mediante programación (continuación)

Nombre de propiedad	Aplicación cliente o de servicio	Valor administrativo	Valor programático	Descripción
MaxConcurrentCalls	Servicio	El rango es 1 - 2 147 483 647 16 es el valor predeterminado	El rango es 1 - 2 147 483 647 16 es el valor predeterminado	Esta propiedad solo se utiliza si la característica de entrega garantizada está habilitada (para obtener más información sobre la entrega garantizada, consulte "Entrega garantizada del canal personalizado WCF" en la página 1304). Especifica el número máximo de operaciones simultáneas que puede haber en curso al mismo tiempo para el punto final dado. Es necesario ser precavido al cambiar este valor. Cada operación simultánea requiere recursos adicionales, especialmente una nueva instancia del canal personalizado y las hebras asociadas de la agrupación de hebras para accionar las solicitudes. Una sobreasignación puede ser contraproducente y afectar de forma grave al rendimiento. Debe realizarse una configuración adecuada de la agrupación de hebras para dar soporte a esta propiedad.

Creación y alojamiento de servicios para WCF

Descripción general de los servicios Microsoft Windows Communication Foundation (WCF) que explica cómo crear y configurar servicios WCF.

El canal personalizado de IBM MQ para WCF y los servicios WCF que lo usan se pueden alojar empleando los métodos siguientes:

- Autoalojamiento
- Servicio de Windows

El canal personalizado de IBM MQ para WCF no se puede alojar en Windows Process Activation Service.

Los temas siguientes proporcionan algunos ejemplos sencillos de autoalojamiento para ilustrar los pasos necesarios. La documentación en línea de Microsoft WCF, que contiene información adicional y los últimos detalles, se puede encontrar en el sitio web de Microsoft MSDN en https://msdn.microsoft.com.

Creación de aplicaciones de servicio WCF utilizando el método 1: autoalojamiento administrativo usando un archivo de configuración de aplicación

Tras haber creado un archivo de configuración de aplicación, abra una instancia del servicio y añada el código especificado a la aplicación.

Antes de empezar

Cree o edite un archivo de configuración de aplicación para el servicio, tal y como se describe en "Creación de un canal personalizado WCF administrativamente suministrando la información de enlace y punto final en un archivo de configuración de aplicaciones" en la página 1309

Acerca de esta tarea

1. Cree una instancia del servicio y ábrala en el host de servicios. El tipo de servicio tiene que coincidir con el tipo de servicio especificado en el archivo de configuración del servicio.
2. Añada el código siguiente a la aplicación:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

Creación de aplicaciones de servicio WCF utilizando el método 2: autoalojamiento programático directamente desde la aplicación

Añada las propiedades de enlace, cree el host de servicio con una instancia de la clase de servicio necesaria y abra el servicio.

Antes de empezar

1. Añada una referencia al archivo `IBM.XMS.WCF.dll` del canal personalizado en el proyecto. `IBM.XMS.WCF.dll` se encuentra en *`WMQInstallDir`*`\bin` donde *WMQInstallDir* es el directorio en el que está instalado IBM MQ.
2. Añada una sentencia *using* al espacio de nombres `IBM.XMS.WCF`, por ejemplo: `using IBM.XMS.WCF`
3. Cree una instancia del elemento de enlace y punto final del canal, tal como se describe en: "Creación de un canal personalizado WCF suministrando la información de enlace y punto final mediante programación" en la página 1311

Acerca de esta tarea

Si se necesitan cambios en las propiedades de enlace del canal, siga estos pasos:

1. Añada las propiedades de enlace a `transportBindingElement`, tal como se muestra en el ejemplo siguiente:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Cree el host de servicio con una instancia de la clase de servicio necesaria:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Abra el servicio:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

Exposición de metadatos utilizando un punto final HTTP

Instrucciones para exponer los metadatos de un servicio que está configurado para utilizar el canal personalizado de IBM MQ para WCF.

Acerca de esta tarea

Si hay que exponer los metadatos de servicios (para que herramientas como `svcutil` puedan acceder directamente desde el servicio en ejecución y no desde un archivo WSDL fuera de línea, por ejemplo), hay que hacerlo exponiendo dichos metadatos con un punto final HTTP. Se pueden utilizar los pasos siguientes para añadir este punto final adicional.

1. Añada la dirección base de donde haya que exponer los metadatos a ServiceHost, por ejemplo:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Añada este código a ServiceHost antes de abrir el servicio:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

Resultados

Ahora los metadatos están disponibles en la dirección siguiente: `http://localhost:8000/MyService`

Creación de aplicaciones cliente para WCF

Descripción general de la generación y creación de aplicaciones cliente de Microsoft Windows Communication Foundation (WCF).

Se puede crear una aplicación cliente para un servicio WCF; las aplicaciones cliente suelen generarse con la Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) para crear los archivos de configuración y proxy necesarios que puede utilizar directamente la aplicación.

Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con metadatos de un servicio en ejecución

Instrucciones para utilizar la herramienta de Microsoft svcutil.exe para generar un cliente para un un servicio que está configurado para utilizar el canal personalizado de IBM MQ para WCF.

Antes de empezar

Hay tres requisitos previos al uso de la herramienta svcutil para crear la configuración y los archivos proxy necesarios que puede usar directamente la aplicación:

- El servicio WCF tiene que estar en ejecución antes de iniciarse la herramienta svcutil.
- El servicio WCF debe exponer sus metadatos utilizando un puerto HTTP además de las referencias de punto final de canal personalizado de IBM MQ para generar un cliente directamente desde un servicio en ejecución.
- El canal personalizado tiene que estar registrado en los datos de configuración de svcutil.

Acerca de esta tarea

Los pasos siguientes explican cómo generar un cliente para un servicio que está configurado para utilizar un canal personalizado de IBM MQ, pero que también expone sus metadatos durante el tiempo de ejecución a través de un puerto HTTP separado:

1. Inicie el servicio WCF (el servicio tiene que estar ejecutando antes de iniciar la herramienta svcutil).

2. Añada los detalles del archivo de configuración `svcutil.exe` de la raíz de la instalación, en el archivo de configuración de svcutil activo, normalmente `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`, de forma que svcutil reconozca el canal personalizado de IBM MQ.
3. Ejecute svcutil por línea de comandos, por ejemplo:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Copie los archivos `app.config` y `YourService.cs` generados en el proyecto de cliente de Microsoft Visual Studio.

Qué hacer a continuación

Si los metadatos de servicios no se pueden recuperar directamente, se puede usar svcutil para generar los archivos de cliente a partir de un wsdl en su lugar. Puede obtener información adicional consultando: "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con WSDL" en la página 1320

Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con WSDL

Instrucciones para la generación de clientes WCF a partir de WSDL si los metadatos del servicio no están disponibles.

Si los metadatos del servicio no se pueden recuperar directamente para generar un cliente a partir de los metadatos de un servicio en ejecución, svcutil se puede utilizar para generar los archivos de cliente a partir de WSDL en su lugar. Las modificaciones siguientes se deben realizar en el WSDL para especificar que se va a utilizar el canal personalizado IBM MQ.

1. Añada las siguientes definiciones de espacio de nombres e información de política:

```
<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-
utility-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. Modifique la sección de enlaces para que haga referencia a la nueva sección de política y elimine cualquier definición `transport` del elemento binding subyacente:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Ejecute svcutil por línea de comandos, por ejemplo:

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

Creación de aplicaciones de cliente WCF utilizando un proxy de cliente con un archivo de configuración de aplicaciones

Antes de empezar

Crear o editar un archivo de configuración de aplicación para el cliente, tal como se describe en: "Creación de un canal personalizado WCF administrativamente suministrando la información de enlace y punto final en un archivo de configuración de aplicaciones" en la página 1309

Acerca de esta tarea

Crear una instancia y abrir una instancia del proxy de cliente. El parámetro pasado al proxy generado debe ser el mismo que el nombre de punto final especificado en el archivo de configuración del cliente, por ejemplo, `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();
            }
```

Creación de aplicaciones de cliente WCF utilizando un proxy de cliente con configuración programática.

Antes de empezar

1. Añada una referencia al archivo `IBM.XMS.WCF.dll` del canal personalizado en el proyecto. `IBM.XMS.WCF.dll` se encuentra en el directorio *`WMQInstallDir`*`\bin` donde *WMQInstallDir* es el directorio en el que se ha instalado IBM MQ.
2. Añada una sentencia *using* al espacio de nombres `IBM.XMS.WCF`, por ejemplo: `using IBM.XMS.WCF`
3. Cree una instancia del elemento de enlace y punto final del canal, según se describe en: "Creación de un canal personalizado WCF suministrando la información de enlace y punto final mediante programación" en la página 1311

Acerca de esta tarea

Si hacen falta cambios a las propiedades de enlace del canal, realice los pasos siguientes.

1. Añada las propiedades de enlace a `transportBindingElement` según se muestra en la figura siguiente:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. Cree el proxy de cliente tal como se muestra en la figura siguiente, donde *binding* y *endpoint address* son la dirección de enlace y punto final configurada en el paso 1 y pasadas en:

```
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();
           }
```

Utilización de los ejemplos de WCF

Los ejemplos de Windows Communication Foundation (WCF) son ejemplos sencillos de cómo se puede usar un canal personalizado de IBM MQ.

Para crear los proyectos de ejemplo, se necesitan el SDK de Microsoft.NET 3.5 o Microsoft Visual Studio 2008.

Ejemplo sencillo de WCF servidor y cliente simple unidireccional

Este ejemplo ilustra el canal personalizado de IBM MQ que se está utilizando para iniciar un servicio Windows Communication Foundation (WCF) desde un cliente WCF que utiliza una forma de canal unidireccional.

Acerca de esta tarea

El servicio implementa un único método que saca una cadena por consola. El cliente se ha geerado con la herramienta `svcutil` que recupera los metadatos del servicio de un punto final HTTP expuesto aparte, tal y como se describe en "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con metadatos de un servicio en ejecución" en la página 1319

El ejemplo se ha configurado con nombres de recursos concretos, tal y como se describe en el procedimiento siguiente. Si debe cambiar los nombres de recurso, también debe cambiar el valor correspondiente en la aplicación cliente en el archivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` y en la aplicación de servicio en el archivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación para IBM MQ. Para obtener más información sobre cómo dar formato al URI de punto final JMS, consulte *Transporte IBM MQ para SOAP* en la documentación del producto IBM MQ. Si tiene que modificar la solución de ejemplo y el código fuente , necesitará un IDE como, por ejemplo, Microsoft Visual Studio 8 o superior.

Procedimiento

1. Cree un gestor de colas llamado *QM1*
2. Cree un destino de cola llamado *SampleQ*
3. Inicie el servicio para que el escucha esté a la espera de mensajes: ejecute el archivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación para IBM MQ.

4. Ejecute el cliente una vez: Ejecute el archivo *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

   La aplicación cliente repite cinco veces el envío de cinco mensajes a *SampleQ*

Resultados

La aplicación de servicio obtiene mensajes de *SampleQ* y muestra `Hello World` en la pantalla cinco veces.

Qué hacer a continuación

Ejemplo sencillo WCF de cliente de solicitud-respuesta y servidor

Este ejemplo ilustra el canal personalizado IBM MQ que se está utilizando para iniciar un servicio Windows Communication Foundation (WCF) desde un cliente WCF utilizando una forma de canal de solicitud-respuesta.

Acerca de esta tarea

Este servicio proporciona algunos métodos de calculadora simples para añadir y restar dos números, y devolver el resultado. El cliente se ha geerado con la herramienta `svcutil` que recupera los metadatos del servicio de un punto final HTTP expuesto aparte, tal y como se describe en "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con metadatos de un servicio en ejecución" en la página 1319

El ejemplo se ha configurado con nombres de recursos concretos, tal y como se describe en el procedimiento siguiente. Si necesita cambiar los nombres de recurso, también tendrá que cambiar el valor correspondiente en la aplicación cliente en el archivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config`, y en la aplicación de servicio en el archivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación para IBM MQ. Para obtener más información sobre cómo dar formato al URI de punto final JMS, consulte *Transporte IBM MQ para SOAP* en la documentación del producto IBM MQ. Si tiene que modificar la solución de ejemplo y el código fuente , necesitará un IDE como, por ejemplo, Microsoft Visual Studio 8 o superior.

Procedimiento

1. Cree un gestor de colas llamado *QM1*
2. Cree un destino de cola llamado *SampleQ*
3. Cree un destino de cola llamado *SampleReplyQ*
4. Inicie el servicio para que el escucha esté a la espera de mensajes: ejecute el archivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación para IBM MQ.
5. Ejecute una vez el cliente: Ejecute el archivo *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

Resultados

Cuando haya ejecutado el cliente, se iniciará el proceso siguiente y se repetirá cuatro veces para que se envíe un total de cinco mensajes en cada sentido:

1. El cliente coloca un mensaje de solicitud en *SampleQ* y espera una respuesta.
2. El servicio obtiene el mensaje de solicitud de *SampleQ*.
3. El servicio añade y resta algunos valores contenidos en el mensaje.

4. A continuación, el servicio coloca el resultado en un mensaje en *SampleReplyQ* y espera a que el cliente coloquque un mensaje nuevo.
5. El cliente obtiene el mensaje de *SampleReplyQ* y saca resultado por pantalla.

Qué hacer a continuación

Ejemplo de cliente WCF para un servicio .NET alojado por IBM MQ

Las aplicaciones cliente de ejemplo y las aplicaciones de proxy de servicio de ejemplo se proporcionan para .NET y, también, para Java. Los ejemplos se basan en un servicio de cotización en bolsa que recibe una petición de cotización en bolsa y proporciona dicha cotización.

Antes de empezar

El ejemplo requiere que el entorno que aloja el servicio de .NET SOAP sobre JMS esté correctamente instalado y configurado en IBM MQ y sea accesible desde un gestor de colas local.

Cuando el entorno que aloja el servicio de .NET SOAP sobre JMS esté correctamente instalado y configurado en IBM MQ y sea accesible desde un gestor de colas local, habrá que realizar pasos de configuración adicionales.

1. Establezca la variable de entorno **WMQSOAP_HOME** en el directorio de instalación IBM MQ , por ejemplo: `C:\Archivos de programa\IBM\MQ`
2. Asegúrese de que el compilador Java `javac` está disponible y en la variable PATH.
3. Copie el archivo `axis.jar` del directorio `prereqs/axis` de la imagen de instalación en el directorio de producción IBM MQ , por ejemplo: `C:\Archivos de programa\IBM\MQ\java\lib\soap`
4. Añada a la PATH: *MQ_INSTALLATION_PATH*`\Java\lib` donde *MQ_INSTALLATION_PATH* representa el directorio en el que está instalado IBM MQ, por ejemplo: `C:\Archivos de programa\IBM\MQ`
5. Asegúrese de que la ubicación de .NET se especifica correctamente en *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ, por ejemplo: `C:\Archivos de programa\IBM\MQ`. La ubicación de .NET se puede especificar, por ejemplo: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

cuando haya terminado los pasos anteriores, ejecute y pruebe el servicio:

1. Vaya hasta el directorio de trabajo de SOAP sobre JMS.
2. Especifique uno de los comandos siguientes para ejecutar la prueba de verificación y dejar el escucha del servicio ejecutando:
   - Para .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ .
   - Para AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ .

El argumento `hold` mantiene a los escuchas en ejecución una vez finalizada la prueba.

Si se notifican errores durante esta configuración, se pueden eliminar todos los cambios para poder volver a iniciar el procedimiento de la siguiente manera:

1. Suprima el directorio generado de SOAP sobre JMS.
2. Suprima el gestor de colas.

Acerca de esta tarea

Este ejemplo ilustra una conexión desde un cliente WCF con el ejemplo de servicio .NET SOAP sobre JMS que se proporciona en IBM MQ usando un ajuste de canal unidireccional. El servicio implementa un ejemplo de cotización en bolsa simple, que genera una cadena de texto por consola.

El cliente se ha generado usando WSDL para generar archivos de cliente, tal y como se describe en "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con WSDL" en la página 1320

El ejemplo se ha configurado con nombres de recursos concretos, tal y como se describe en el procedimiento siguiente. Si tiene que cambiar los nombres de recurso, también debe cambiar el valor correspondiente en la aplicación cliente en el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\app.config`, y en la aplicación de servicio en el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl`, donde *MQ_INSTALLATION_PATH* representa el directorio de instalación para IBM MQ. Para obtener más información sobre cómo formatear el URI de punto final JMS , consulte *IBM MQ Transport for SOAP* en la documentación del producto IBM MQ .

Procedimiento

Ejecute el cliente una vez: ejecute el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe`, donde *MQ_INSTALLATION_PATH* representa el directorio de instalación para IBM MQ.

La aplicación cliente itera cinco veces enviando cinco mensajes a la cola de ejemplo.

Resultados

La aplicación de servicio obtiene los mensajes de la cola de ejemplo y saca `Hello World` cinco veces por pantalla.

Ejemplo de cliente WCF de un servicio Java Axis alojado en IBM MQ

Las aplicaciones cliente de ejemplo y las aplicaciones de proxy de servicio de ejemplo se proporcionan para Java y, también, para .NET. Los ejemplos se basan en un servicio de cotización en bolsa que recibe una petición de cotización en bolsa y proporciona dicha cotización.

Antes de empezar

Este ejemplo requiere que el entorno que aloja el servicio de .NET SOAP sobre JMS esté correctamente instalado y configurado en IBM MQ y sea accesible desde un gestor de colas local.

Cuando el entorno que aloja el servicio de .NET SOAP sobre JMS esté correctamente instalado y configurado en IBM MQ y sea accesible desde un gestor de colas local, habrá que realizar pasos de configuración adicionales.

1. Establezca la variable de entorno **WMQSOAP_HOME** en el directorio de instalación IBM MQ , por ejemplo: `C:\Archivos de programa\IBM\MQ`
2. Asegúrese de que el compilador Java `javac` está disponible y en la variable PATH.
3. Copie el archivo `axis.jar` del directorio `prereqs/axis` de la imagen de instalación en el directorio de instalación de IBM MQ .
4. Añada a la PATH: *MQ_INSTALLATION_PATH*`\Java\lib` donde *MQ_INSTALLATION_PATH* representa el directorio en el que está instalado IBM MQ, por ejemplo: `C:\Archivos de programa\IBM\MQ`
5. Asegúrese de que la ubicación de .NET se especifica correctamente en *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ, por ejemplo: `C:\Archivos de programa\IBM\MQ`. La ubicación de .NET se puede especificar, por ejemplo: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

cuando haya terminado los pasos anteriores, ejecute y pruebe el servicio:

1. Vaya hasta el directorio de trabajo de SOAP sobre JMS.
2. Especifique uno de los comandos siguientes para ejecutar la prueba de verificación y dejar el escucha del servicio ejecutando:

- Para .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ .
- Para AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado IBM MQ .

El argumento hold mantiene a los escuchas en ejecución una vez finalizada la prueba.

Si se notifican errores durante esta configuración, se pueden eliminar todos los cambios para que se vuelva a iniciar el procedimiento de la siguiente manera:

1. Suprima el directorio generado de SOAP sobre JMS.
2. Suprima el gestor de colas.

Acerca de esta tarea

El ejemplo ilustra una conexión desde un cliente WCF al ejemplo de servicio Java Axis SOAP sobre JMS que se proporciona en IBM MQ usando un ajuste de canal unidireccional. El servicio implementa un ejemplo de cotización en bolsa simple, que genera una cadena de texto en un archivo que se guarda en el directorio actual.

El cliente se ha generado usando WSDL para generar archivos de cliente, tal y como se describe en "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con WSDL" en la página 1320

El ejemplo se ha configurado con nombres de recursos concretos, tal y como se describe en este párrafo. Si necesita cambiar los nombres de recurso, también debe cambiar el valor correspondiente en la aplicación cliente en el archivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\app.config y en la aplicación de servicio en el archivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl , donde *MQ_INSTALLATION_PATH* representa el directorio de instalación para IBM MQ.

Procedimiento

Ejecute el cliente una vez: ejecute el archivo *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe, donde *MQ_INSTALLATION_PATH* representa el directorio de instalación para IBM MQ.

La aplicación cliente itera cinco veces enviando cinco mensajes a la cola de ejemplo.

Resultados

La aplicación de servicio obtiene los mensajes de la cola de ejemplo y añade Hello World cinco veces a un archivo en el directorio actual.

Ejemplo de cliente WCF para el servicio Java alojado por WebSphere Application Server

Se proporcionan aplicaciones cliente de ejemplo y aplicaciones proxy de servicio de ejemplo para WebSphere Application Server 6. También se proporciona un servicio de solicitud y respuesta.

Antes de empezar

Este ejemplo requiere que se utilice la configuración de IBM MQ siguiente:

Tabla 195. Configuración de IBM MQ necesaria

Objeto	**Nombre necesario**
Gestor de colas	QM1
Cola local	HelloWorld

Tabla 195. Configuración de IBM MQ necesaria (continuación)	
Objeto	**Nombre necesario**
Cola local	HelloWorldReply

Este ejemplo también requiere que un entorno de alojamiento de WebSphere Application Server 6 esté correctamente instalado y configurado. WebSphere Application Server 6 utiliza una conexión de modalidad de enlaces para conectarse a IBM MQ de forma predeterminada. Por lo tanto, WebSphere Application Server 6 debe estar instalado en la misma máquina que el gestor de colas.

Una vez configurado el entorno WAS, se deben realizar los pasos de configuración adicionales siguientes:

1. Cree los siguientes objetos JNDI en el repositorio JNDI de WebSphere Application Server:
   a. Un destino de cola JMS con el nombre `HelloWorld`
      - Establezca el nombre JNDI en `jms/HelloWorld`
      - Establezca el nombre de cola en `HelloWorld`
   b. Una fábrica de conexiones de cola JMS con el nombre `HelloWorldQCF`
      - Establezca el nombre JNDI en `jms/HelloWorldQCF`
      - Establezca el nombre del gestor de colas en `QM1`
   c. Una fábrica de conexiones de cola JMS con el nombre `WebServicesReplyQCF`
      - Establezca el nombre JNDI en `jms/WebServicesReplyQCF`
      - Establezca el nombre del gestor de colas en `QM1`
2. Cree un puerto de escucha de mensajes con el nombre `HelloWorldPort` en WebSphere Application Server con la configuración siguiente:
   - Establezca el nombre JNDI de la fábrica de conexiones en `jms/HelloWorldQCF`
   - Establezca el nombre JNDI de destino en `jms/HelloWorld`
3. Instale la aplicación `HelloWorldEJBEAR.ear` de servicio web en WebSphere Application Server de este modo:
   a. Pulse **Aplicaciones** > **Nueva aplicación** > **Nueva aplicación empresarial**.
   b. Vaya a *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.
   c. No modifique ninguna opción predeterminada del asistente y reinicie el servidor de aplicaciones una vez instalada la aplicación.

Cuando se haya completado la configuración de WAS, pruebe el servicio ejecutándolo una vez:

1. Vaya hasta el directorio de trabajo de SOAP sobre JMS.
2. Escriba este mandato para ejecutar el ejemplo: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

Acerca de esta tarea

El ejemplo muestra una conexión de un cliente WCF con el servicio de ejemplo WebSphere Application Server SOAP sobre JMS proporcionado en los ejemplos WCF incluidos en IBM MQ, utilizando una forma de canal de solicitud-respuesta. El flujo de mensajes se realiza entre WCF y WebSphere Application Server utilizando las colas de IBM MQ. El servicio implementa el método `HelloWorld(...)` , que toma una serie y devuelve un saludo al cliente.

El cliente se ha generado con la herramienta svcutil para recuperar los metadatos del servicio desde un punto final HTTP, expuesto de forma separada, como se describe en la sección "Generación de un proxy cliente WCF y de archivos de configuración de aplicación con la herramienta svcutil con metadatos de un servicio en ejecución" en la página 1319

El ejemplo se ha configurado con nombres de recursos concretos, tal y como se describe en el procedimiento siguiente. Si necesita cambiar los nombres de recursos, también debe cambiar el valor correspondiente de la aplicación de cliente en el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\app.config` y de la aplicación de servicio en el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

El servicio y el cliente están basados en el servicio y cliente descritos en el artículo de IBM Developer: *Crear un servicio web de JMS utilizando SOAP a través de JMS y WebSphere Studio*. Para obtener más información sobre el desarrollo de servicios web SOAP sobre JMS que son compatibles con el canal personalizado WCF de IBM MQ , consulte https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Procedimiento

Ejecutar el cliente una vez: Ejecute el archivo *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe`, donde *MQ_INSTALLATION_PATH* es el directorio de instalación de IBM MQ.

La aplicación de cliente inicia los dos métodos de servicio al mismo tiempo, enviando dos mensajes a la cola de ejemplo.

Resultados

La aplicación de servicio obtiene los mensajes de la cola de ejemplo y proporciona una respuesta a la llamada al método `HelloWorld(...)` que la aplicación cliente genera en la consola.

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los Estados Unidos.

Es posible que IBM no ofrezca los productos, servicios o las características que se tratan en este documento en otros países. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Las referencias a programas, productos o servicios de IBM no pretenden establecer ni implicar que sólo puedan utilizarse dichos productos, programas o servicios de IBM. En su lugar podrá utilizarse cualquier producto, programa o servicio equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio no IBM.

IBM puede tener patentes o solicitudes de patentes pendientes que cubran el tema principal descrito en este documento. El suministro de este documento no le otorga ninguna licencia sobre estas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Para consultas sobre licencias relacionadas con información de doble byte (DBCS), póngase en contacto con el Departamento de propiedad intelectual de IBM de su país o envíe las consultas por escrito a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokio 103-8510, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones contradigan la legislación vigente: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN NINGÚN TIPO DE GARANTÍA, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUYENDO, PERO SIN LIMITARSE A, LAS GARANTÍAS IMPLÍCITAS DE NO INCUMPLIMIENTO, COMERCIABILIDAD O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas legislaciones no contemplan la exclusión de garantías, ni implícitas ni explícitas, en determinadas transacciones, por lo que puede haber usuarios a los que no les afecte dicha norma.

Esta información puede contener imprecisiones técnicas o errores tipográficos. La información aquí contenida está sometida a cambios periódicos; tales cambios se irán incorporando en nuevas ediciones de la publicación. IBM puede realizar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento sin previo aviso.

Cualquier referencia en esta información a sitios web que no son de IBM se realiza por razones prácticas y de ninguna manera sirve como un respaldo de dichos sitios web. Los materiales de dichos sitios web no forman parte de este producto de IBM y la utilización de los mismos será por cuenta y riesgo del usuario.

IBM puede utilizar o distribuir cualquier información que el usuario le proporcione del modo que considere apropiado sin incurrir por ello en ninguna obligación con respeto al usuario.

Los titulares de licencias de este programa que deseen información del mismo con el fin de permitir: (i) el intercambio de información entre los programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N

© Copyright IBM Corp. 2007, 2024

Rochester, MN 55901
U.S.A.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluyendo, en algunos casos, el pago de una cantidad.

El programa bajo licencia que se describe en esta información y todo el material bajo licencia disponible para el mismo lo proporciona IBM bajo los términos del Acuerdo de cliente de IBM, el Acuerdo de licencia de programas internacional de IBM o cualquier acuerdo equivalente entre las partes.

Los datos de rendimiento incluidos en este documento se han obtenido en un entorno controlado. Por consiguiente, los resultados obtenidos en otros entornos operativos pueden variar de manera significativa. Es posible que algunas mediciones se hayan realizado en sistemas en nivel de desarrollo y no existe ninguna garantía de que estas mediciones serán las mismas en sistemas disponibles generalmente. Además, es posible que algunas mediciones se hayan estimado mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a su entorno específico.

La información relativa a productos que no son de IBM se obtuvo de los proveedores de esos productos, sus anuncios publicados u otras fuentes de disponibilidad pública. IBM no ha comprobado estos productos y no puede confirmar la precisión de su rendimiento, compatibilidad o alguna reclamación relacionada con productos que no sean de IBM. Todas las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de dichos productos.

Todas las declaraciones relacionadas con una futura intención o tendencia de IBM están sujetas a cambios o se pueden retirar sin previo aviso y sólo representan metas y objetivos.

Este documento contiene ejemplos de datos e informes que se utilizan diariamente en la actividad de la empresa. Para ilustrar los ejemplos de la forma más completa posible, éstos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con los nombres y direcciones utilizados por una empresa real es puramente casual.

LICENCIA DE DERECHOS DE AUTOR:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente que ilustran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar ninguna cuota a IBM para fines de desarrollo, uso, marketing o distribución de programas de aplicación que se ajusten a la interfaz de programación de aplicaciones para la plataforma operativa para la que se han escrito los programas de ejemplo. Los ejemplos no se han probado minuciosamente bajo todas las condiciones. IBM, por tanto, no puede garantizar la fiabilidad, servicio o funciones de estos programas.

Puede que si visualiza esta información en copia software, las fotografías e ilustraciones a color no aparezcan.

Información acerca de las interfaces de programación

La información de interfaz de programación, si se proporciona, está pensada para ayudarle a crear software de aplicación para su uso con este programa.

Este manual contiene información sobre las interfaces de programación previstas que permiten al cliente escribir programas para obtener los servicios de WebSphere MQ.

Sin embargo, esta información puede contener también información de diagnóstico, modificación y ajustes. La información de diagnóstico, modificación y ajustes se proporciona para ayudarle a depurar el software de aplicación.

Importante: No utilice esta información de diagnóstico, modificación y ajuste como interfaz de programación porque está sujeta a cambios.

Marcas registradas

IBM, el logotipo de IBM , ibm.com, son marcas registradas de IBM Corporation, registradas en muchas jurisdicciones de todo el mundo. Hay disponible una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas.

Microsoft y Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o otros países.

UNIX es una marca registrada de Open Group en Estados Unidos y en otros países.

Linux es una marca registrada de Linus Torvalds en Estados Unidos y en otros países.

Este producto incluye software desarrollado por Eclipse Project (https://www.eclipse.org/).

Java y todas las marcas registradas y logotipos son marcas registradas de Oracle o sus afiliados.

Número Pieza:

(1P) P/N: