7.5

Desarrollo de aplicaciones para IBM WebSphere MQ

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información en “Avisos” en la página 1143.

Esta edición se aplica a la versión 7 release 5 de IBM® WebSphere 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.

Contenido

Desarrollo de aplicaciones

IBM WebSphere MQ proporciona varias formas en las que puede desarrollar aplicaciones para enviar y recibir mensajes que necesita para dar soporte a los procesos de negocio. También puede desarrollar aplicaciones para gestionar los gestores de colas y recursos relacionados.

Antes de desarrollar aplicaciones para IBM WebSphere MQ, asegúrese de que está familiarizado con los conceptos de IBM WebSphere MQ Visión general técnicaIBM WebSphere MQ Visión general técnica.

Puede desarrollar aplicaciones para IBM WebSphere MQ en varios lenguajes de programación diferentes. Para obtener información sobre los lenguajes de programación soportados y sus características, consulte “Decidir qué lenguaje de programación utilizar” en la página 80.

Consulte las secciones siguientes para ver los tipos de aplicaciones que puede escribir para IBM WebSphere MQ en distintas plataformas.

Tipos de aplicación que puede escribir para IBM WebSphere MQ

Esta información trata sobre los tipos de aplicaciones que se pueden escribir en IBM WebSphere MQ.

Los productos IBM WebSphere MQ son gestores de colas y habilitadores de aplicaciones. Dan soporte a la interfaz de cola de mensajes (MQI) de IBM a través de la cual los programas pueden colocar mensajes en una cola y obtener mensajes de una cola.

Con IBM WebSphere MQ para plataformas noz/OS , puede escribir aplicaciones que:

- 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 WebSphere MQ.
- Utilice la gestión de colas de mensajes desde CICS para TXSeries para AIX, TXSeries para HP-UX, TXSeries para Solaris y TXSeries para aplicaciones de sistemas Windows .
- Utilice la colocación de mensajes en colas desde dentro de Encina para sistemas AIX, HP-UX, Solaris y Windows .
- Utilice la gestión de colas de mensajes desde Tuxedo para los sistemas AIX, AT & T, HP-UX, Solaris y Windows .
- Utilizar IBM WebSphere MQ como gestor de transacciones, coordinando actualizaciones realizadas por gestores de recursos externos dentro de las unidades de trabajo de IBM WebSphere 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.
- Ejecute desde un entorno IBM WebSphere MQ completo o desde un entorno de cliente MQI de IBM WebSphere MQ en las plataformas siguientes:
  - UNIX and Linux® sistemas
  - Windows

Conceptos relacionados

Seguridad

Conceptos de desarrollo de aplicaciones

Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

Antes de empezar a diseñar y escribir las aplicaciones IBM WebSphere MQ , familiarícese con los conceptos básicos de IBM WebSphere MQ , consulte los temas de Visión general técnica. Para obtener información sobre los tipos de aplicación que puede escribir para IBM WebSphere MQ, consulte “Desarrollo de aplicaciones” en la página 7.

Utilice los enlaces siguientes para obtener más información sobre los conceptos de IBM WebSphere MQ específicos del desarrollo de aplicaciones:

- “Mensajes de IBM WebSphere MQ” en la página 9
- Mensajería punto a punto
- Introducción a la mensajería de publicación/suscripción de WebSphere MQ
- “Utilización de la interfaz de cola de mensajes (MQI) en una aplicación cliente” en la página 361
- “Utilización de servicios web en WebSphere MQ” en la página 965
- “Programas de salida de canal para canales de mensajes” en la página 405
- “Escenarios de soporte transaccional” en la página 42

Para poder ejecutar aplicaciones que utilicen la MQI, debe crear determinados objetos de IBM WebSphere MQ. Para obtener más información, consulte “Programas de aplicación que utilizan la MQI” en la página 9.

Conceptos relacionados

“Diseño de aplicaciones IBM WebSphere MQ” en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

“Programas WebSphere MQ de ejemplo” en la página 98
Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

“Escritura de una aplicación de gestión de colas” en la página 199
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 cliente” en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

“Decidir qué lenguaje de programación utilizar” en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

“Utilización de clases de WebSphere MQ para JMS” en la página 729
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) es el proveedor JMS que se proporciona con WebSphere MQ. Además de implementar las interfaces definidas en el paquete javax.jms , WebSphere MQ classes for JMS proporciona dos conjuntos de extensiones para la API JMS.

“Utilización de la interfaz de modelo de objeto de componente (WebSphere MQ Automation Classes for ActiveX)” en la página 1052
Las clases de automatización de WebSphere MQ para ActiveX (MQAX) son componentes ActiveX que proporcionan clases que puede utilizar en la aplicación para acceder a WebSphere MQ.

“Utilización de clases de WebSphere MQ para Java” en la página 666
WebSphere MQ classes for Java le permite utilizar WebSphere MQ en un entorno Java. Una aplicación Java puede utilizar clases WebSphere MQ para Java o clases WebSphere MQ para JMS para acceder a recursos WebSphere MQ .

“Utilización de .NET” en la página 571
WebSphere MQ classes for .NET permite a un programa escrito en la infraestructura de programación .NET conectarse a WebSphere MQ como un cliente MQI de WebSphere MQ o conectarse directamente a un servidor WebSphere MQ .

“Utilización de C++” en la página 641
WebSphere MQ proporciona clases C++ equivalentes a los objetos WebSphere MQ y algunas clases adicionales equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

“Creación de una aplicación IBM WebSphere MQ” en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

Programas de aplicación que utilizan la MQI

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

En la Figura 1 en la página 9, 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.

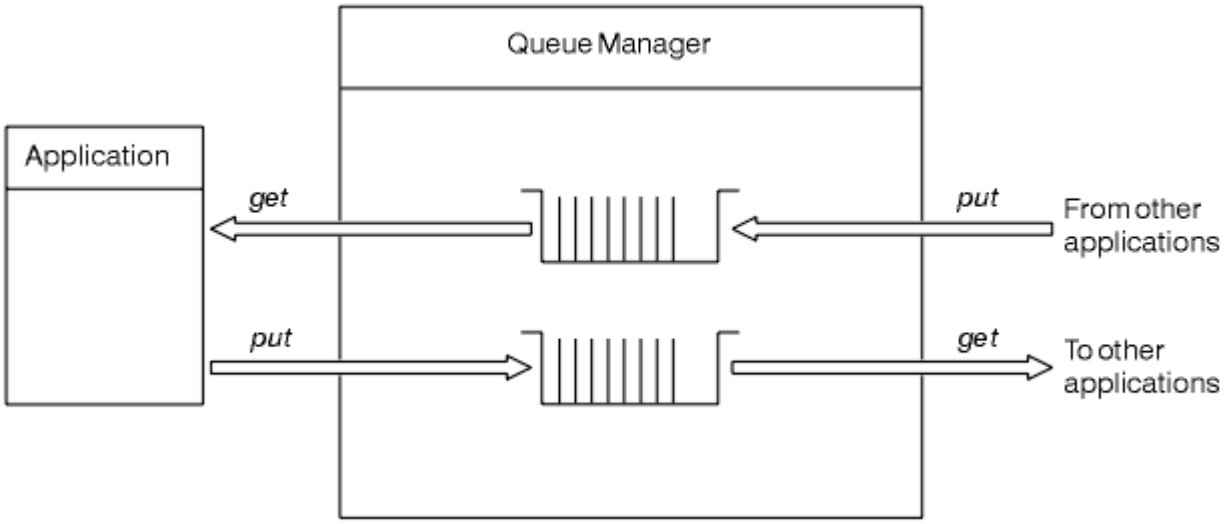


Figura 1. Colas, mensajes y aplicaciones

Mientras que las aplicaciones pueden colocar mensajes en colas locales o remotas (utilizando MQPUT), sólo pueden obtener mensajes directamente de 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 WebSphere MQ. Consulte “Creación de una aplicación IBM WebSphere MQ” en la página 438.
- 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 9.

Mensajes de IBM WebSphere MQ

Esta información presenta el concepto de mensaje IBM WebSphere MQ , las partes del mensaje y el descriptor de mensaje.

Los mensajes de IBM WebSphere MQ constan de dos partes:

- Propiedades del mensaje
- Datos de aplicación

La Figura 2 en la página 10 representa un mensaje y muestra cómo se divide lógicamente en las propiedades de los mensajes y los datos de aplicación.

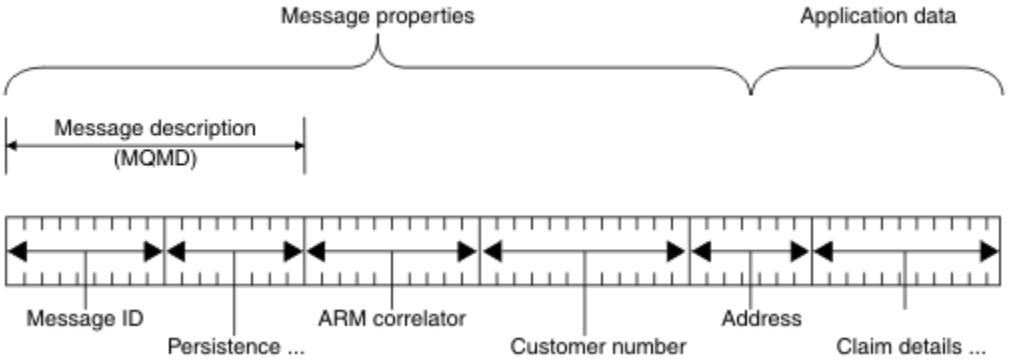


Figura 2. Representación de un mensaje

Los datos de aplicación transportados en un mensaje de WebSphere MQ no los cambia un gestor de colas a menos que se lleve a cabo la conversión de datos en el mismo. Además, WebSphere MQ no pone ninguna restricción sobre el contenido de estos datos. La longitud de los datos de cada mensaje no puede superar el valor del atributo *MaxMsgLength* tanto de la cola como del gestor de colas.

En WebSphere MQ para AIX, WebSphere MQ para HP-UX, WebSphere MQ para Linux, WebSphere MQ para Solaris, y WebSphere MQ para Windows, el valor predeterminado de *MaxMsgLength* es 100 MB (104 857 600 bytes).

Haga que los mensajes sean ligeramente más cortos que el valor del atributo *MaxMsgLength* en algunas circunstancias. Consulte “Los datos del mensaje” en la página 235 para obtener más información.

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.

Consulte “Contexto de mensaje” en la página 39 para obtener una descripción de cómo utilizar los campos dentro del MQMD que contienen información sobre el origen del mensaje.

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 36). Es el mismo que el descriptor de mensaje de la versión 1, pero tiene campos adicionales. Estos se describen en MQMDE-Extensión de descriptor de mensaje.

Tipos de mensajes

Hay cuatro tipos de mensajes definidos por IBM WebSphere 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 *MsgId* o el *CorrelId* del mensaje original se copien 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 *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*) e 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 respuesta en el campo *ReplyToQ* ; de lo contrario, MQPUT o MQPUT1 del mensaje original fallará 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 los mensajes de informe que se crean para el mensaje:

- Puede solicitar que el *MsgId* o el *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 que el *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 los campos *MsgId* y *CorrelId* en el mensaje de respuesta o informe adecuadamente.

  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 estas aplicaciones normalmente necesitan reenviar el mensaje a la aplicación de servidor con los campos *MsgId*, *CorrelId*y *Report* sin cambios. Esto permite a la aplicación de servidor copiar el *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 WebSphere MQ define un rango de códigos de comentarios para que los utilicen 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).

Informes y mensajes segmentados

No soportado en WebSphere MQ para z/OS .

Si un mensaje está segmentado (consulte “Segmentación de mensajes” en la página 269 para obtener una descripción de los mensajes segmentados) y solicita que se generen informes, es posible que reciba más informes de los que habría recibido si el mensaje no se hubiera segmentado.

Para informes generados por WebSphere 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 la aplicación genera informes, copie siempre las cabeceras de WebSphere MQ que están presentes al principio de los datos del mensaje original en los datos del mensaje de 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 WebSphere MQ que deben copiarse consultando los nombres de formato sucesivos, empezando por MQMD y continuando por las cabeceras presentes. Los siguientes nombres `Format` indican estas cabeceras de WebSphere 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 otras cabeceras de WebSphere MQ aparece en la misma posición. La longitud de la cabecera está contenida 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 de mensaje originales excluyendo las longitudes de cualquier cabecera WebSphere MQ que encuentre.

Recuperación de informes

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

Un MQGET con MQGMO_COMPLETE_MSG se satisface cuando hay suficientes mensajes de informe (de un solo tipo, por ejemplo COA, y con el mismo *GroupId*) en la cola para representar un mensaje original completo. Esto es cierto incluso si 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 ese mensaje de informe, incluso si los datos en sí no están presentes.

Puede utilizar esta técnica incluso si hay varios tipos de informe diferentes presentes en la cola (por ejemplo, COA y COD), porque un MQGET con MQGMO_COMPLETE_MSG vuelve a ensamblar los mensajes de informe sólo si tienen el mismo código *Feedback* . Sin embargo, normalmente no puede utilizar esta técnica para los informes de excepción, porque, en general, estos tienen códigos *Feedback* diferentes.

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, es posible que obtenga distintos códigos *Feedback* para distintos segmentos y que obtenga 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* en

el 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 genera un informe que da soporte a la segmentación, pero se recibe en un gestor de colas que *no* da soporte a la segmentación, la estructura MQMDE (que identifica *Offset* y *OriginalLength* representados por el informe) siempre se incluye en los datos del 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 de 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 dicho 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 empiecen por MQ para estos.

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 espera la 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 coloca 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* en el 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 de WebSphere MQ , por ejemplo, formatos de mandatos programables

  WebSphere MQ utiliza mensajes de formato de mandato programable para mensajes y sucesos de administración (el nombre de formato utilizado es MQFMT_ADMIN en este caso). 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 424. 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 WebSphere 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 códigos mixta, WebSphere MQ utiliza la parte SBCS de la página de códigos mixta como CCSID del gestor de colas.

Para la conversión de datos general, si el sistema operativo subyacente da soporte a páginas de códigos DBCS, WebSphere MQ puede utilizarlo.

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 424 para obtener información acerca de cómo invocar las salidas de conversión de datos.

Prioridades de mensajes

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 de la cola se almacenan en la secuencia FIFO (primero en entrar, primero en salir) o en FIFO dentro de la secuencia 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 221), 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 dicho gestor de colas. No puede modificar el valor de este atributo. En WebSphere MQ, el atributo tiene el valor 9; puede crear mensajes que tengan prioridades entre 0 (el más bajo) y 9 (el más alto).

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. También facilitan la comunicación entre WebSphere MQ y las aplicaciones 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 propiedades de mensaje en WebSphere MQ imita el uso de propiedades en JMS. Esto significa que puede establecer propiedades en una aplicación JMS y recuperarlas en una aplicación WebSphere MQ de procedimiento, o al revés. Para que una propiedad esté disponible para una aplicación JMS, asígnele el prefijo "usr"; a continuación, estará disponible (sin el prefijo) como una propiedad de usuario de mensaje JMS. Por ejemplo, la propiedad WebSphere MQ *usr.myproperty* (una serie de caracteres) es accesible para una aplicación JMS utilizando la llamada JMS `message.getStringProperty('myproperty')`. Tenga en cuenta que las aplicaciones JMS no pueden acceder a las propiedades con el prefijo "usr" si contienen dos o más 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 WebSphere MQ añadiendo el "usr." al nombre de propiedad consultado en una llamada MQINQMP.

Propiedades y longitud de un mensaje

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

Cuando se fluye a un gestor de colas anterior a la versión 7.0 del producto, las propiedades del mensaje, excepto las del descriptor de mensaje, cuentan para la longitud del mensaje. Por lo tanto, debe aumentar el valor del atributo *MaxMsgLength* de los canales que van a un sistema anterior a la versión 7.0 según sea necesario, para compensar el hecho de que se puedan enviar más datos para cada mensaje. De forma alternativa, se puede bajar el valor *MaxMsgLength* de la cola o del gestor de colas para que el nivel global de los datos que se envían por el sistema siga siendo el mismo.

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.

Puesto que no hay una 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 WebSphere 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, WebSphereMQ 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 el inicio. Las reglas para los identificadores Java son iguales 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 WebSphere 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 fácil de las propiedades; por ejemplo, una aplicación puede consultar todas las propiedades de mensaje a partir de "`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"-reservado para que lo utilicen las clases WebSphere MQ para JMS.
   - "usr.JMS"-no válido.

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

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

Propiedad	Sinónimo de
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 a una aplicación MQI acceder a las propiedades JMS de forma similar a las clases de WebSphere MQ para la aplicación cliente JMS. De estas propiedades, sólo JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupID y JMSXGroupSeq se pueden establecer utilizando la interfaz MQI.

Tenga en cuenta que las propiedades JMS_IBM_ * disponibles en WebSphere MQ classes for JMS no están disponibles utilizando 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. Estos son los nombres de las palabras clave SQL utilizadas en las series de selección.
3. Un nombre de propiedad que empieza por "mq " en cualquier mezcla de minúsculas o mayúsculas y que no empiece por "mq_usr" sólo 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 establece la propiedad "a.b" y, a continuación, la propiedad "a.b.c", no está claro si en la jerarquía "b" contiene un valor u otra agrupación lógica. Una jerarquía de este tipo es de "contenido mixto" y este no está soportado. No está permitido definir una propiedad que produzca contenido contenido mixto.

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

- Los nombres de propiedad se validan al establecer una propiedad utilizando la llamada MQSETMP: Establecer propiedad de mensaje , si se ha solicitado la validación al crear el manejador de mensajes. Si se realiza un intento de validar una propiedad y falla debido a un error en la especificación del nombre de propiedad, el código de terminación es MQCC_FAILED con la razón:
  - 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. < Campo>	< 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 con un MQMDE que es respetado por el gestor de colas, se puede acceder a los campos MQMDE utilizando la notación Root.MQMD.<Field> descrita. En este caso, los campos MQMDE se tratan como una parte lógica de MQMD desde la perspectiva de las propiedades. Consulte la sección "MQMDE especificado en llamadas MQPUT y MQPUT1 " en 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 serie de bytes no es un tipo de datos de propiedad válido en JMS o XMS. Se recomienda no utilizar propiedades de serie 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 realizar selecciones utilizando las llamadas MQSUB y MQOPEN.

La estructura *SelectionString* se utiliza para pasar una serie 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.

Si se especifica un CCSID para el que no hay ninguna conversión Unicode soportada de WebSphere MQ , se produce 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 para el campo *VSCCSID* es MQCCSI_APPL, que indica que el CCSID de la serie de selección es igual al CCSID del gestor de colas, o al CCSID del cliente si está 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 serie de selección sólo está limitada por lo que puede describir el campo MQCHARV *VSLength*.

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 24 muestra el proceso de selección utilizando la llamada MQSUB.

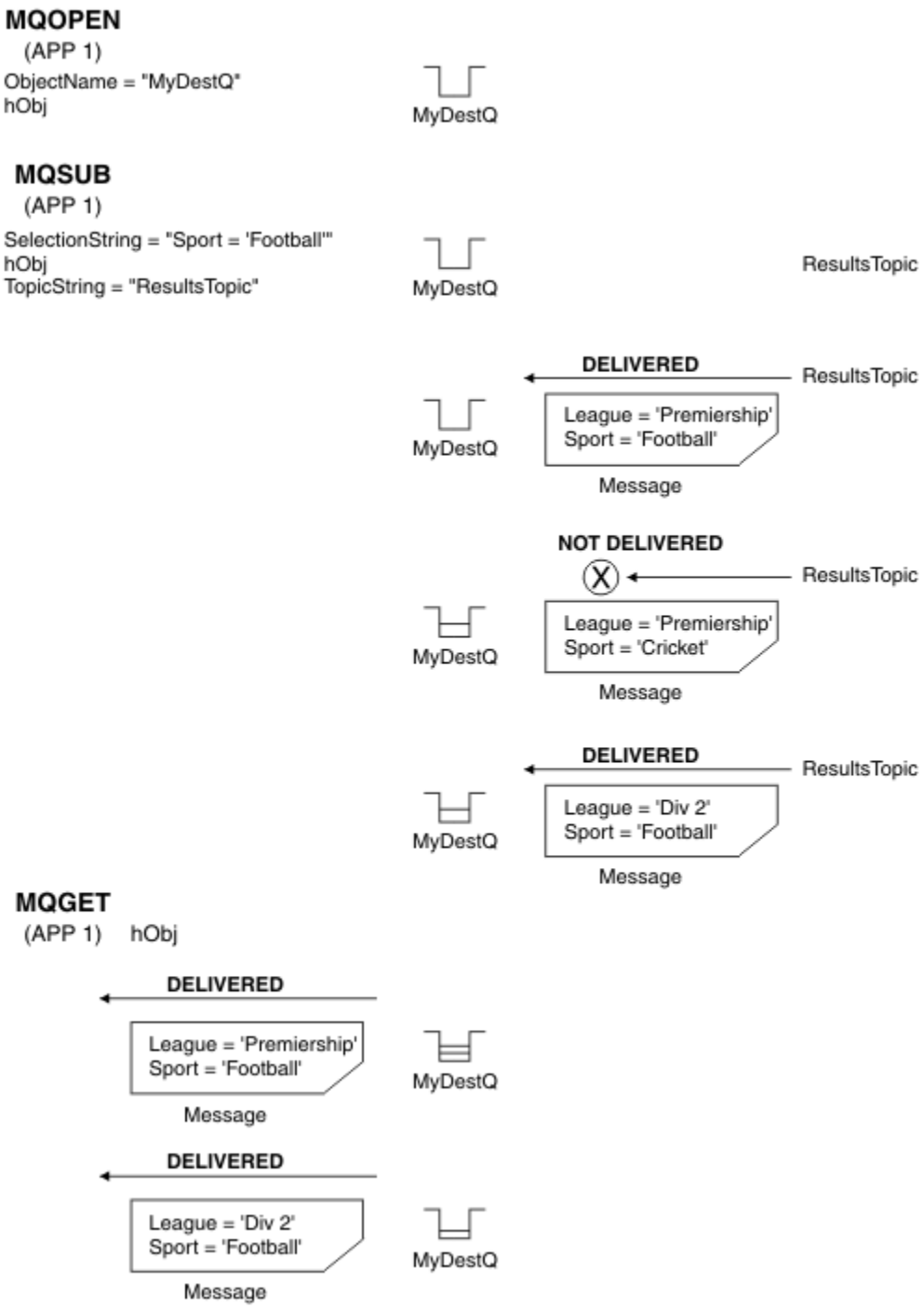


Figura 3. Selección con una llamada MQSUB

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 25 muestra el proceso de selección utilizando la llamada MQOPEN.

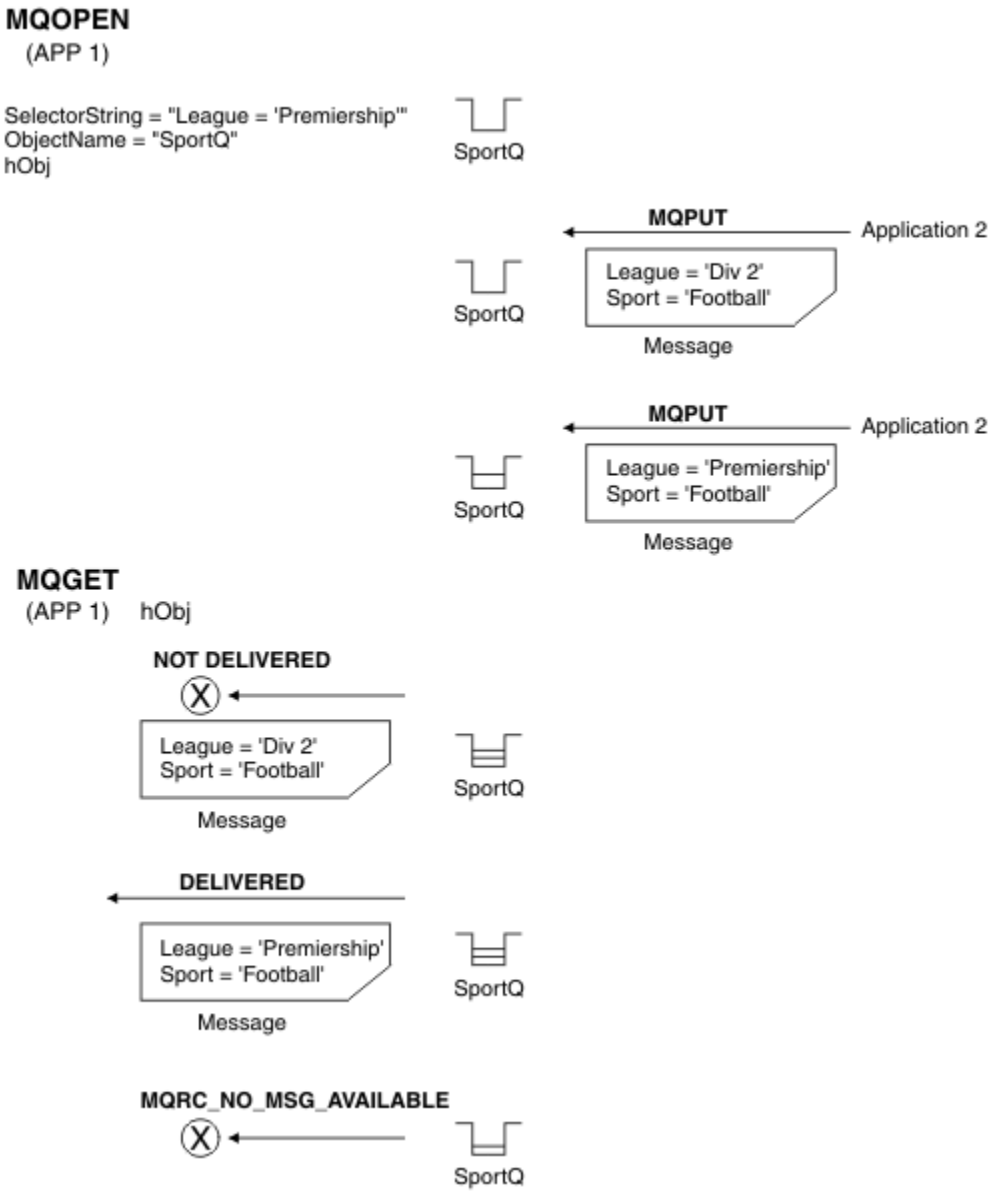


Figura 4. Selección con la llamada MQOPEN

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.

Comportamiento de la selección
Visión general del comportamiento de selección de IBM WebSphere 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 que una serie de selección se resuelva en TRUE o FALSE antes de que tenga lugar cualquier coincidencia 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 puede seguir satisfaciendo una serie de selección como `'Color IS NULL'`, donde `'Color'` no existe como una 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 WebSphere 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 = ...` o `JMSMessageID = ...` y haga referencia solo a una propiedad.

Este método ofrece una mejora significativa en el rendimiento de la selección en JMSCorrelationID y ofrece una mejora de rendimiento marginal en JMSMessageID.

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 WebSphere MQ protege el sistema no procesando selectores excesivamente complejos que podrían provocar una escasez de recursos del sistema. La protección puede producirse después de aproximadamente 100 pruebas en algunas plataformas, por lo que los selectores que se acercan a ese número de componentes pueden ver anomalías. Se recomienda que el uso de selectores con muchos componentes se prueben a fondo en las plataformas adecuadas 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

Sintaxis del selector de mensajes
Un selector de mensajes de WebSphere MQ es una serie con sintaxis que se basa en un subconjunto de la sintaxis de expresión condicional SQL92 .

Selección del contenido de un mensaje
Es posible suscribirse basándose en una selección de contenido de carga útil de mensaje (también conocido como filtrado de contenido), pero la decisión sobre qué mensajes deben entregarse a dicha suscripción no la puede realizar directamente WebSphere MQ; en su lugar, se necesita un proveedor de selección de mensajes ampliado, por ejemplo, IBM Integration Bus, para procesar los mensajes.

Sintaxis del selector de mensajes
Un selector de mensajes de WebSphere MQ es una serie con sintaxis que 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.

WebSphere 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.

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'. Al igual que los literales de serie Java, estos utilizan 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`. Los ejemplos son `"0x2F1C"` o `"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 no se considera endianness)

– Los números hexadecimales empiezan con un cero, seguido de un `x`en 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, 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. WebSphere 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.`, `-95.7`o `+6.2`. WebSphere MQ da soporte a números en el rango de `-1.797693134862315E+308` a `1.797693134862315E+308`.

  El significado debe ir después de 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 19 y “Restricciones para nombres de propiedad” en la página 20 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 propiedad JMS que se correlacionan con nombres de propiedad o nombres de campo MQMD también son identificadores válidos en una serie de selección. WebSphere MQ correlaciona el campo JMS reconocido y los nombres de propiedad con los valores de propiedad de mensaje. Consulte “Selectores de mensajes en JMS” en la página 826 para obtener más información. Como ejemplo, la serie de selección `"JMSPriority >="` selecciona en la propiedad `Pri` que se encuentra en la carpeta `jms` del mensaje actual.

- Desbordamiento/subdesbordamiento:

  Para los números numéricos decimales y aproximados, lo siguiente no está definido:

  – 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 está definida por las reglas de promoción numérica 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 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. 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. 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
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tabla 3. 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

Comportamiento de la selección
Visión general del comportamiento de selección de IBM WebSphere MQ .

Selección del contenido de un mensaje
Es posible suscribirse basándose en una selección de contenido de carga útil de mensaje (también conocido como filtrado de contenido), pero la decisión sobre qué mensajes deben entregarse a dicha suscripción no la puede realizar directamente WebSphere MQ; en su lugar, se necesita un proveedor de selección de mensajes ampliado, por ejemplo, IBM Integration Bus, para procesar los mensajes.

“Propiedades del mensaje” en la página 18
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. También facilitan la comunicación entre WebSphere MQ y las aplicaciones 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 equivalencia se prueba utilizando un único carácter 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 que `a != 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 para conjunción/disyunción lógica; por ejemplo, `a && b` debe especificarse como `a AND b`.
- Los caracteres comodín `*` y `?` no son sinónimos para `%` y `_`.
- Los selectores que contienen expresiones compuestas 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 debe escribirse 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` sólo existe para dar soporte a los casos `IS NULL` y `IS NOT NULL`.

Conceptos relacionados

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

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 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.

Conceptos relacionados

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.

Selección del contenido de un mensaje

Es posible suscribirse basándose en una selección de contenido de carga útil de mensaje (también conocido como filtrado de contenido), pero la decisión sobre qué mensajes deben entregarse a dicha suscripción no la puede realizar directamente WebSphere MQ; en su lugar, se necesita un proveedor de selección de mensajes ampliado, por ejemplo, IBM Integration Bus, para procesar los mensajes.

Cuando una aplicación publica en una serie de tema, donde uno o más suscriptores tienen una serie de selección que se selecciona en el contenido del mensaje, WebSphere MQ solicitará que el proveedor de selección de mensajes ampliados analice la publicación e informe a WebSphere MQ si la publicación coincide con los criterios de selección especificados 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.

Conceptos relacionados

Comportamiento de la selección
Visión general del comportamiento de selección de IBM WebSphere MQ .

Sintaxis del selector de mensajes
Un selector de mensajes de WebSphere MQ es una serie con sintaxis que se basa en un subconjunto de la sintaxis de expresión condicional SQL92 .

Consumo asíncrono de mensajes IBM WebSphere 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 los entornos de aplicación más sencillos, 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 35 muestra una aplicación multihebra que consume mensajes de dos colas. El ejemplo muestra todos los mensajes que se entregan a una sola función.

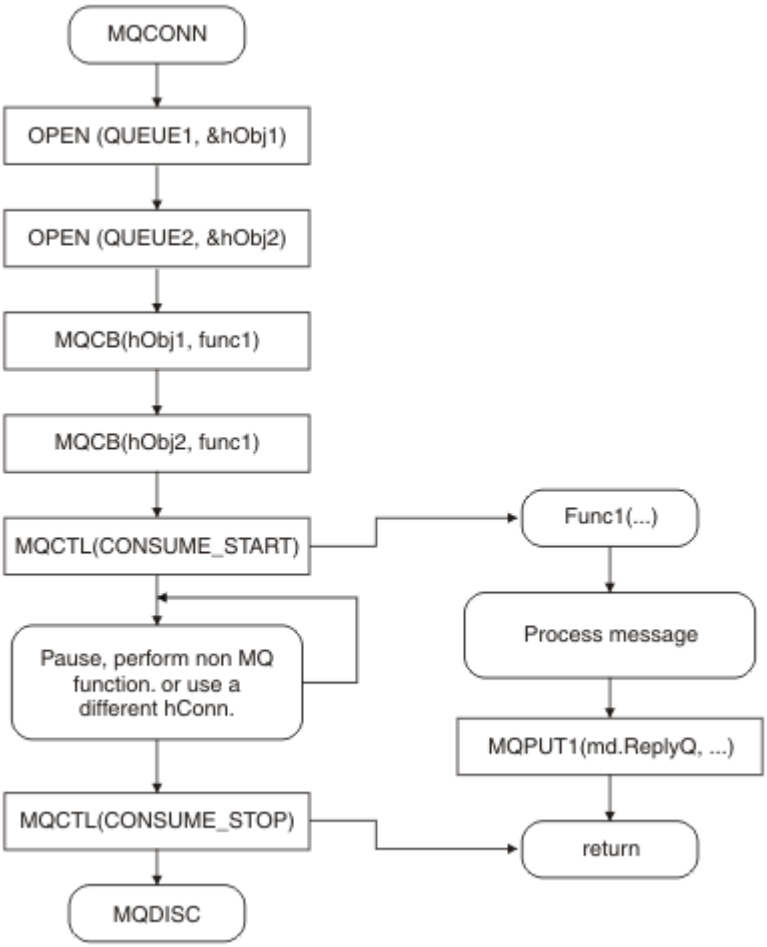


Figura 5. Aplicación controlada por mensajes estándar que consume de dos colas

Figura 6 en la página 36 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.

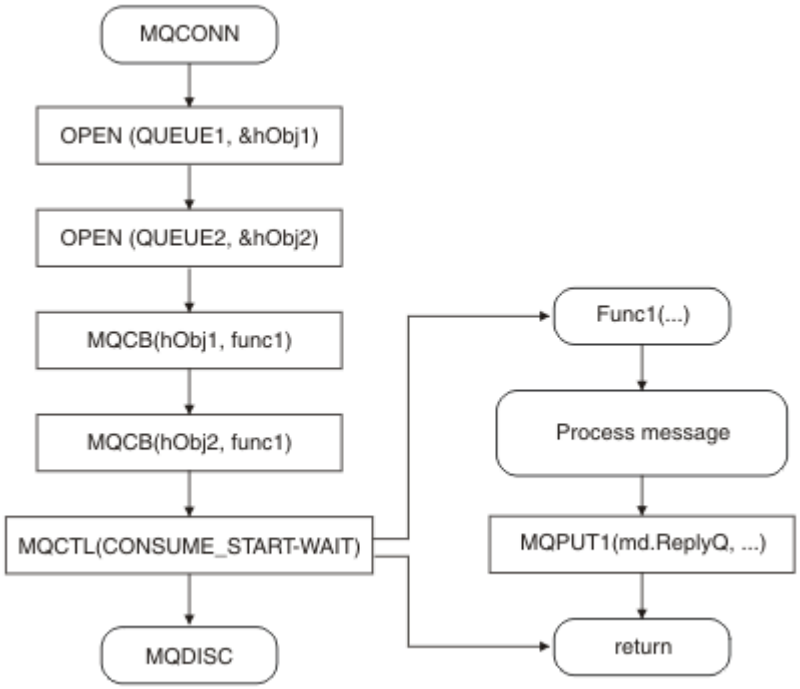


Figura 6. Aplicación controlada por mensajes de hebra única que consume de dos colas

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 251). En plataformas que no son z/OS, un concepto relacionado, "Segmentación de mensajes" en la página 269 permite que los mensajes grandes se dividan en segmentos más pequeños. No puede utilizar mensajes agrupados o segmentados al colocar en un tema.

La jerarquía de un grupo es la siguiente:

Grupo
Es el nivel más alto de la jerarquía y se identifica mediante un *GroupId*. Consta de uno o más 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 36 se muestra un grupo de mensajes lógicos:

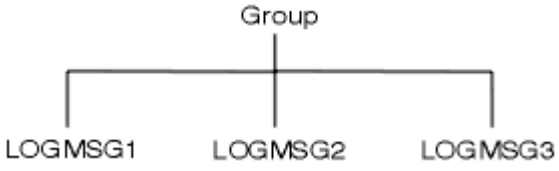


Figura 7. Grupo de mensajes lógicos

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 dentro de un grupo se identifican mediante los campos *GroupId* y *MsgSeqNumber* . El *MsgSeqNumber* empieza en 1 para el primer mensaje dentro de 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. Cada mensaje es lógicamente un mensaje independiente, y solo los campos *GroupId* y *MsgSeqNumber* del MQMD necesitan tener alguna 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 269.

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* empieza en cero para el primer segmento dentro de un mensaje.

Cada segmento consiste en un mensaje físico que podría pertenecer a un grupo (en la Figura 8 en la página 37 se muestra un ejemplo de los mensajes que hay un grupo). Un segmento forma parte lógicamente de un único mensaje, por lo que sólo los campos *MsgId*, *Offset*y *SegmentFlag* del MQMD deben diferir entre segmentos separados 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 37 se muestra un grupo de mensajes lógicos, algunos de los cuales están segmentados:

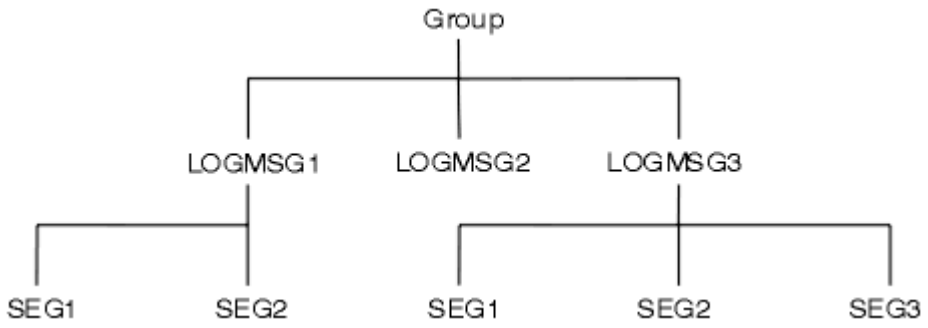


Figura 8. Mensajes segmentados

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

Para obtener una descripción de los mensajes lógicos y físicos, consulte “Ordenación lógica y física” en la página 251. Para obtener más información sobre la segmentación de mensajes, consulte “Segmentación de mensajes” en la página 269.

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.

Al crear un mensaje, si inicializa el descriptor de mensaje (MQMD) utilizando los valores predeterminados, la persistencia del mensaje se toma 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 330.
- En IBM WebSphere MQ en sistemas UNIX , IBM WebSphere MQ en sistemas Linux , y IBM WebSphere MQ para Windows, 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.

Consulte “Manejo de errores de programa” en la página 559 para obtener más información.

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

En WebSphere MQ para WebSphere MQ para Windows, WebSphere MQ en sistemas UNIX , WebSphere MQ en sistemas Linux el recuento de restituciones siempre sobrevive a los reinicios del gestor de colas. Cualquier cambio en el atributo *HardenGetBackout* se ignora.

Para obtener más información sobre cómo confirmar y restituir mensajes, consulte “Confirmación y restitución de unidades de trabajo” en la página 330.

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).

Utilizando la estructura MQMD, especifique el nombre de la cola a la que desea enviar los mensajes de respuesta y de informe, en el campo *ReplyToQ* . Especifique el nombre del gestor de colas que es propietario de la cola de respuestas 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 de 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; de lo contrario, este campo no se cambia.

ReplyToQMgr
Si *ReplyToQ* es una definición local de una cola remota, el campo *ReplyToQMgr* se establece en el nombre del gestor de colas propietario de la cola remota; de lo contrario, el campo *ReplyToQMgr* se establece en el nombre del gestor de colas al que está 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 562).

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” en la página 236.

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; la forma 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 ha determinado de la aplicación que ha colocado el mensaje.
- Las aplicaciones pueden utilizar el campo *ApplIdentityData* para cualquier información adicional que deseen incluir sobre el usuario (por ejemplo, una contraseña cifrada).

Un identificador de seguridad (SID) de sistemas Windows se almacena en el campo *AccountingToken* cuando se crea un mensaje en WebSphere MQ para 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 rellena 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 de *contexto de origen* describe la aplicación que coloca el mensaje en la cola en la que el mensaje está almacenado *actualmente* . El descriptor de mensaje contiene los campos siguientes para poder obtener información del contexto de origen:

PutApplType	El tipo de aplicación que coloca el mensaje (por ejemplo, una transacción CICS).
PutApplName	El nombre de la aplicación que ha colocado el mensaje (por ejemplo, el nombre de un trabajo o transacción).
PutDate	La fecha en la que se ha colocado el mensaje en la cola.
PutTime	Hora a la que se ha colocado el mensaje en la cola.
ApplOriginData	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 WebSphere MQ

Esta información proporciona detalles sobre objetos de WebSphere 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 que WebSphere 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 utiliza mandatos WebSphere MQ para definir, modificar 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" en la página 236
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 227
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.

Preparación y ejecución de aplicaciones de Microsoft Transaction Server

Para preparar una aplicación MTS para que se ejecute como una aplicación cliente MQI de WebSphere MQ , siga estas instrucciones según convenga para su entorno.

Para obtener información general sobre cómo desarrollar aplicaciones de Microsoft Transaction Server (MTS) que acceden a los recursos de WebSphere MQ , consulte la sección sobre MTS en el Centro de ayuda de WebSphere MQ .

Para preparar una aplicación MTS para que se ejecute como una aplicación cliente MQI de WebSphere MQ , 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 469, pero enlace el componente con la biblioteca mqicxa.lib en vez de mqic.lib.
- Si el componente utiliza las clases C++ de WebSphere MQ , siga las instrucciones de "Creación de programas C++ en Windows" en la página 665 pero enlace el componente con 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 de Visual Basic en Windows" en la página 473, pero cuando defina el proyecto Visual Basic, escriba `MqType=3` en el campo **Argumentos de compilación condicional**,
- Si el componente utiliza WebSphere MQ Automation Classes for ActiveX (MQAX), defina una variable de entorno, GMQ_MQ_LIB, con el valor `mqic32xa.dll` .

  Puede definir la variable de entorno dentro de la aplicación o definirla de forma que su ámbito sea todo el sistema. Sin embargo, la definición a nivel de sistema puede hacer que cualquier aplicación MQAX existente que no defina la variable de entorno desde dentro, se comporte de forma incorrecta.

Utilización de IBM WebSphere MQ con WebSphere Application Server

Utilice este tema para comprender el uso de IBM WebSphere MQ con WebSphere Application Server.

Las aplicaciones escritas en Java que se ejecutan en WebSphere Application Server pueden utilizar la especificación JMS (Java Messaging Service) para realizar la mensajería. La mensajería punto a punto en este entorno puede ser proporcionada por un gestor de colas de IBM WebSphere MQ

Una ventaja de utilizar un gestor de colas IBM WebSphere MQ para proporcionar la mensajería punto a punto es que la conexión de aplicaciones JMS puede participar plenamente en la funcionalidad de una red IBM WebSphere MQ , lo que permite a las aplicaciones intercambiar mensajes con gestores de colas que se ejecutan en una multitud de 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. Si el gestor de colas no es local para la aplicación, la *Conexión de cliente* debe estar instalada para permitir que la aplicación se conecte a un gestor de colas que se ejecuta en otra máquina o imagen.

De forma predeterminada, los mensajes JMS contenidos en colas IBM WebSphere MQ utilizan una cabecera MQRFH2 para contener parte de la información de cabecera de mensaje JMS. Muchas aplicaciones IBM WebSphere MQ heredadas no pueden procesar mensajes con estas cabeceras y requieren sus propias cabeceras características, por ejemplo, MQCIH para CICS Bridge o MQWIH para aplicaciones de flujo de trabajo de IBM WebSphere MQ . Para obtener más detalles sobre estas consideraciones especiales, consulte “Correlación de mensajes JMS con mensajes WebSphere MQ” en la página 829.

Escenarios de soporte transaccional

Mediante el soporte transaccional, puede habilitar sus aplicaciones para trabajar de forma fiable con bases de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Este apartado introduce el soporte transaccional. El trabajo necesario para permitir que las aplicaciones utilicen IBM WebSphere MQ con un producto de base de datos abarca las áreas de programación de aplicaciones y administración del sistema. Utilice la información aquí junto con “Confirmación y restitución de unidades de trabajo” en la página 330.

Empezamos por introducir las unidades de trabajo que forman las transacciones y, a continuación, describimos las formas en que se habilita IBM WebSphere MQ para coordinar las transacciones con las bases de datos.

Conceptos relacionados

“Introducción a las unidades de trabajo” en la página 42
Este tema presenta y define los conceptos generales de unidad de trabajo, confirmación, restitución y punto de sincronización. También contiene dos escenarios que ilustran unidades de trabajo globales.

IBM WebSphere MQ y HP NontStop TMF

Introducción a las unidades de trabajo

Este tema presenta y define los conceptos generales de unidad de trabajo, confirmación, restitución y punto de sincronización. También contiene dos escenarios que ilustran unidades de trabajo globales.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Cuando un programa coloca mensajes en colas dentro de una unidad de trabajo, dichos mensajes se hacen visibles a otros programas sólo 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 decide no hacer permanente la operación de transferencia, puede *restituir* la unidad de trabajo. Cuando un programa realiza una restitución, WebSphere MQ restaura las colas eliminando los mensajes que dicha unidad de trabajo ha colocado en las colas.

Asimismo, cuando un programa obtiene mensajes de una o más colas dentro de una unidad de trabajo, dichos mensajes permanecen en las colas hasta que el programa confirma la unidad de trabajo, pero no pueden ser recuperados por otros programas. Los mensajes se suprimen permanentemente de las colas cuando el programa confirma la unidad de trabajo. Si el programa restituye la unidad de trabajo, WebSphere MQ restaura las colas haciendo que los mensajes estén disponibles para que los recuperen otros programas.

La decisión de confirmar o restituir las modificaciones se toma, en el caso más sencillo, al final de una tarea. No obstante, puede ser más útil para una aplicación sincronizar las modificaciones de datos en otros puntos lógicos de una tarea. Estos puntos lógicos se denominan puntos de sincronización y el período que dura el 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.

Con WebSphere MQ, tenemos que distinguir entre las unidades de trabajo *local* y *global* :

Unidades de trabajo locales
Son aquellas en las que las únicas acciones son transferir y obtener de colas de WebSphere MQ , y la coordinación de cada unidad de trabajo se proporciona dentro del gestor de colas utilizando un proceso de *confirmación en una sola fase* .

Utilice unidades de trabajo locales cuando los únicos recursos que se deben actualizar son las colas gestionadas por un único gestor de colas WebSphere MQ . Las actualizaciones se confirman utilizando el verbo MQCMIT o se restituyen utilizando MQBACK.

No hay ninguna tarea de administración del sistema, distinta de la gestión del registro, implicada en la utilización de unidades de trabajo locales. En sus aplicaciones, donde utiliza llamadas MQPUT y MQGET con MQCMIT y MQBACK, intente utilizar las opciones MQPMO_SYNCPOINT y MQGMO_SYNCPOINT. (Para obtener información sobre la gestión de registros, consulte Gestión de archivos de registro .)

Unidades de trabajo globales
Son aquellas en las que también se actualizan otros recursos, como tablas de una base de datos relacional. Cuando esté implicado más de un *gestor de recursos*, necesitará un software *gestor de transacciones* que utilice un proceso de *confirmación en dos fases* para coordinar la unidad de trabajo global.

Utilice unidades de trabajo globales cuando también necesite incluir actualizaciones en el software del gestor de bases de datos relacionales, como por ejemplo Db2, Oracle, Sybasey Informix.

Existen varios escenarios posibles para el uso de unidades de trabajo globales. A continuación se muestran dos escenarios documentados:

1. En el primero, el gestor de colas actúa él mismo como gestor de transacciones. En este escenario, los verbos MQI controlan las unidades de trabajo globales; se inician en aplicaciones utilizando el verbo MQBEGIN y luego se confirman utilizando MQCMIT o se restituyen utilizando MQBACK.
2. En el segundo, el rol de gestor de transacciones lo realiza otro software, como TXSeries, Encinao Tuxedo. En este escenario, se utiliza una API proporcionada por el software del gestor de transacciones para controlar la unidad de trabajo (por ejemplo, EXEC CICS SYNCPOINT for TXSeries).

Las secciones siguientes describen todos los pasos necesarios para utilizar unidades de trabajo globales, organizados por los dos escenarios:

- “Escenario 1: El gestor de colas realiza la coordinación” en la página 44
- “Escenario 2: Otro software proporciona la coordinación” en la página 71

Escenario 1: El gestor de colas realiza la coordinación

En el escenario 1, el gestor de colas actúa como el gestor de transacciones. En este escenario, los verbos MQI controlan las unidades de trabajo globales; se inician en aplicaciones utilizando el verbo MQBEGIN y luego se confirman utilizando MQCMIT o se restituyen utilizando MQBACK.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Nivel de aislamiento

En IBM WebSphere MQ, es posible que un mensaje esté visible en una cola antes de la actualización de una base de datos, dependiendo del diseño del aislamiento de transacción implementado en la base de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Cuando un gestor de colas de IBM WebSphere MQ está trabajando como gestor de transacciones XA, para coordinar las actualizaciones a los gestores de recursos XA, se sigue el siguiente protocolo de confirmación:

1. Preparar todos los gestores de recursos XA.
2. Confirme el gestor de recursos del gestor de colas IBM WebSphere MQ .
3. Confirmar otros gestores de recursos.

Entre el paso 2 y el paso 3, es posible que una aplicación vea un mensaje confirmado en la cola, pero la fila correspondiente en la base de datos no refleja este mensaje.

Esto no es un problema si la base de datos está configurada de forma que las llamadas de la API de la base de datos de la aplicación, esperen a que finalicen las actualizaciones pendientes.

Puede resolver esto configurando la base de datos de distinta forma. El tipo de configuración necesaria se denomina "nivel de aislamiento". Para obtener más información sobre los niveles de aislamiento, consulte la documentación de la base de datos. También puede configurar el gestor de colas para confirmar los gestores de recursos en el orden inverso siguiente:

1. Preparar todos los gestores de recursos XA.
2. Confirmar otros gestores de recursos.
3. Confirme el gestor de recursos del gestor de colas IBM WebSphere MQ .

Cuando se cambia el protocolo, el gestor de colas de IBM WebSphere MQ se confirma por última vez, por lo que las aplicaciones que leen mensajes de las colas ven un mensaje sólo después de que se haya completado la actualización de la base de datos correspondiente.

Para configurar que el gestor de colas utilice este protocolo cambiado, establezca la variable de entorno **AMQ_REVERSE_COMMIT_ORDER**.

Establezca esta variable de entorno, en el entorno desde el que **strmqm** se ejecuta para iniciar el gestor de cola. Por ejemplo, ejecute lo siguiente en la shell justo antes de iniciar el gestor de cola:

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

Nota: El establecimiento de esta variable de entorno puede provocar un entrada de registro extra por transacción, por lo que esto tendrá un pequeño impacto en el rendimiento de las transacciones.

Coordinación de bases de datos

Cuando el gestor de colas coordina por sí mismo unidades de trabajo globales, es posible integrar actualizaciones en las bases de datos dentro de unidades de trabajo de trabajo. Es decir, que puede escribirse una aplicación mixta de MQI y SQL y que los verbos MQCMIT y MQBACK pueden utilizarse para confirmar o restituir conjuntamente los cambios efectuados en colas y bases de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

El gestor de colas lleva a cabo este proceso utilizando el protocolo de confirmación en dos fases descrito en la publicación *Proceso de transacciones distribuidas de X/Open: la especificación XA*. Cuando debe confirmarse una unidad de trabajo, el gestor de colas pregunta en primer lugar a cada gestor de bases de datos participante si está preparado para confirmar sus actualizaciones. Sólo si todos los participantes, incluido el propio gestor de colas, están preparados para la confirmación, se confirmarán todas las actualizaciones en la cola y las bases de datos. Si alguno de los participantes no puede preparar sus actualizaciones, la unidad de trabajo se restituirá en vez de confirmarse.

En general, una unidad de trabajo global se implementa en una aplicación a través del siguiente método (en pseudocódigo):

```
MQBEGIN
MQGET (incluye el indicador MQGMO_SYNCPOINT en las opciones de mensaje)
MQPUT (incluye el indicador MQPMO_SYNCPOINT en las opciones de mensaje)
SQL INSERT
MQCMIT
```

El objetivo de MQBEGIN es indicar el principio de una unidad de trabajo global. El objetivo de MQCMIT es indicar el final de la unidad de trabajo global y completarla con todos los gestores de recursos participantes mediante un protocolo de confirmación en dos fases.

Cuando la unidad de trabajo (también conocida como *transacción*) se completa correctamente utilizando MQCMIT, todas las acciones realizadas en dicha unidad de trabajo se hacen permanentes o irreversibles. Si, por alguna razón, la unidad de trabajo no se ejecuta correctamente, todas las acciones se restituyen. No es posible que una acción de una unidad de trabajo se vuelva permanente mientras otra se restituye. Se trata del principio de una unidad de trabajo: o todas las acciones de la unidad de trabajo se vuelven permanentes o ninguna de ellas lo hace.

Nota:

1. El programador de aplicaciones puede obligar a una unidad de trabajo a restituirse mediante una llamada MQBACK. La unidad de trabajo también se puede restituir a través del gestor de colas si la aplicación o la base de datos *falla* antes de realizar la llamada a MQCMIT.
2. Si una aplicación llama a MQDISC sin llamar a MQCMIT, el gestor de colas se comportará como si se hubiese llamado a MQCMIT, y confirma la unidad de trabajo.

Entre MQBEGIN y MQCMIT, el gestor de colas no realiza ninguna llamada a la base de datos para actualizar sus recursos. Es decir, el único modo de cambiar las tablas de una base de datos es mediante su código (por ejemplo, SQL INSERT en el pseudocódigo).

Se proporciona un soporte de recuperación completo por si el gestor de colas pierde contacto con alguno de los gestores de bases de datos durante el protocolo de confirmación. Si un gestor de colas deja de estar disponible mientras está pendiente, es decir, si se le ha llamado para que se confirme pero aún ha de recibir la decisión de confirmación o restitución, el gestor de colas recordará el resultado de la unidad de trabajo hasta que la haya entregado correctamente a la base de datos. Del mismo modo, si el gestor de colas termina con operaciones de confirmación pendientes, dichas operaciones se recordarán aunque se reinicie el gestor de colas. Si una aplicación termina de forma inesperada, la integridad de la unidad de trabajo no se pone en peligro pero el resultado depende de la parte del proceso en que haya terminado la aplicación, como se describe en la Tabla 5 en la página 46.

En las tablas siguientes se resume qué sucede cuando la base de datos o la aplicación falla:

Tabla 4. Qué sucede cuando un servidor de base de datos falla

Aparición de la anomalía	**Resultado**
Antes de que la aplicación llame a MQCMIT.	La unidad de trabajo se restituye.

Tabla 4. Qué sucede cuando un servidor de base de datos falla (continuación)	
Aparición de la anomalía	**Resultado**
Durante la llamada de la aplicación a MQCMIT, **antes** de que todas las bases de datos hayan indicado que están preparadas correctamente.	La unidad de trabajo se restituye con un código de razón de MQRC_BACKED_OUT.
Durante la llamada de la aplicación a MQCMIT, **después** de que todas las bases de datos hayan indicado que están preparadas correctamente, pero antes de que todas hayan indicado que se han confirmado correctamente.	El gestor de colas mantiene la unidad de trabajo en estado de recuperación, con un código de razón de MQRC_OUTCOME_PENDING.
Durante la llamada de la aplicación a MQCMIT, **después** de que todas las bases de datos hayan indicado que se han confirmado correctamente.	La unidad de trabajo se confirma con un código de razón de MQRC_NONE.
Después de que la aplicación llame a MQCMIT.	La unidad de trabajo se confirma con un código de razón de MQRC_NONE.

Tabla 5. Qué sucede cuando un programa de aplicación falla	
Aparición de la anomalía	**Resultado**
Antes de que la aplicación llame a MQCMIT.	La unidad de trabajo se restituye.
Durante la llamada de la aplicación a MQCMIT, **antes** de que el gestor de colas haya recibido la solicitud MQCMIT de la aplicación.	La unidad de trabajo se restituye.
Durante la llamada de la aplicación a MQCMIT, **después** de que el gestor de colas haya recibido la solicitud MQCMIT de la aplicación.	El gestor de colas intenta confirmarse utilizando la confirmación en dos fases (sujeta a productos de bases de datos que ejecuten y confirmen correctamente sus partes de la unidad de trabajo).

En caso de que el código de razón de vuelta de MQCMIT sea MQRC_OUTCOME_PENDING, el gestor de colas recuerda la unidad de trabajo hasta que pueda restablecer el contacto con el servidor de la base de datos, y le dice que confirme su parte de la unidad de trabajo. Consulte “Consideraciones cuando se pierde el contacto con el gestor de recursos XA” en la página 63 para obtener información sobre cómo y cuándo se realiza la recuperación.

El gestor de colas se comunica con los gestores de bases de datos utilizando la interfaz XA, como se explica en la publicación *Proceso de transacciones distribuidas de X/Open: la especificación XA*. Ejemplos de estas llamadas de función son xa_open, xa_start, xa_end, xa_prepare y xa_commit. Se utilizan los términos *gestor de transacciones* y *gestor de recursos* en el mismo sentido en que se utilizan en la especificación XA.

restricciones
Existen restricciones en el soporte de coordinación de la base de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Las restricciones son:

- La capacidad de coordinar actualizaciones de base de datos dentro de unidades de trabajo de WebSphere MQ **no** está soportada en una aplicación cliente MQI. El uso de MQBEGIN en una aplicación de cliente falla. Un programa que llama a MQBEGIN debe ejecutarse como una aplicación de *servidor* en la misma máquina que el gestor de colas.

Nota: Una aplicación de *servidor* es un programa que se ha enlazado con las bibliotecas de servidor de WebSphere MQ necesarias; una aplicación de *cliente* es un programa que se ha enlazado con las bibliotecas de cliente de WebSphere MQ necesarias. Consulte "Creación de aplicaciones para clientes MQI de WebSphere MQ" en la página 366 y "Creación de una aplicación IBM WebSphere MQ" en la página 438 para obtener detalles sobre cómo compilar y enlazar los programas.

- El servidor de bases de datos puede residir en una máquina distinta de la del servidor del gestor de colas, siempre y cuando el cliente de base de datos esté instalado en la misma máquina que el gestor de colas, y soporte esta función. Consulte la documentación del producto de base de datos para determinar si el software cliente se puede utilizar para sistemas de confirmación en dos fases.
- Aunque el gestor de colas se comporte como un gestor de recursos (con objeto de participar en las unidades de trabajo globales del escenario 2), no es posible hacer que un gestor de colas coordine otro gestor de colas dentro de las unidades de trabajo del escenario 1.

Archivos de carga conmutada

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

El archivo de carga conmutada es una biblioteca compartida (una DLL en sistemas Windows) que se carga a través del código en la aplicación de IBM WebSphere MQ y el gestor de colas. Su objeto es simplificar la carga de la biblioteca compartida de cliente de la base de datos, y devolver los punteros a las funciones XA.

Los detalles del archivo de carga conmutada deben especificarse antes de iniciar el gestor de colas. Los detalles se colocan en el archivo qm.ini en los sistemas Windows, UNIX and Linux .

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice IBM WebSphere MQ Explorer para actualizar el archivo qm.ini .
- En todos los demás sistemas, edite el archivo qm.ini directamente.

El código fuente de C para el archivo de carga conmutada se proporciona con la instalación de IBM WebSphere MQ si hay soporte para las unidades de trabajo globales del Escenario 1. El fuente contiene una función denominada MQStart. Cuando se carga el archivo de carga conmutada, el gestor de colas llama a esta función, que devuelve la dirección de una estructura llamada *conmutación XA*.

La estructura de conmutación XA existe en la biblioteca compartida del cliente de base de datos y contiene una serie de punteros de función, tal y como se describe en la Tabla 6 en la página 47:

Tabla 6. Punteros de función de conmutación XA

Nombre del puntero de función	**Función XA**	**Finalidad**
xa_open_entry	xa_open	Conectar con base de datos
xa_close_entry	xa_close	Desconectar de la base de datos
xa_start_entry	xa_start	Iniciar una rama de una unidad de trabajo global
xa_end_entry	xa_end	Suspender una rama de una unidad de trabajo global
xa_rollback_entry	xa_rollback	Deshacer una rama de una unidad de trabajo global
xa_prepare_entry	xa_prepare	Preparar confirmación de una rama de una unidad de trabajo global
xa_commit_entry	xa_commit	Confirmar una rama de una unidad de trabajo global

Tabla 6. Punteros de función de conmutación XA (continuación)		
Nombre del puntero de función	**Función XA**	**Finalidad**
xa_recover_entry	xa_recover	Descubrir en la base de datos si existe una unidad de trabajo pendiente
xa_forget_entry	xa_forget	Permitir que una base de datos olvide una rama de una unidad de trabajo global
xa_complete_entry	xa_complete	Completar una rama de una unidad de trabajo global

Durante la primera llamada MQBEGIN en la aplicación, el código de IBM WebSphere MQ que se ejecuta como parte de MQBEGIN carga el archivo de carga conmutada y llama a la función xa_open en la biblioteca compartida de base de datos. Asimismo, durante el inicio del gestor de colas y en ocasiones posteriores, algunos procesos del gestor de colas cargan el archivo de carga conmutada y llaman a xa_open.

Puede reducir el número de llamadas xa_* utilizando el *registro dinámico*. Para obtener una descripción completa de esta técnica de optimización, consulte “Registro dinámico de XA” en la página 68.

Configuración del sistema para la coordinación de bases de datos

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Existen varias tareas que se deben realizar antes de que un gestor de bases de datos pueda participar en unidades de trabajo globales coordinadas por el gestor de colas. Éstas se describen a continuación:

- “Instalación y configuración del producto de base de datos” en la página 48
- “Creación de archivos de carga conmutada” en la página 49
- “Adición de información de configuración al gestor de colas” en la página 50
- “Escritura y modificación de las aplicaciones” en la página 51
- “Pruebas del sistema” en la página 52

Instalación y configuración del producto de base de datos
Para instalar y configurar el producto de base de datos, consulte la documentación propia del producto. En estos temas de esta sección se describen los problemas generales de configuración y cómo se relacionan con la interoperación entre WebSphere MQ y la base de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Conexiones con bases de datos

Una aplicación que establece una conexión estándar con el gestor de colas se asociará a una hebra en un proceso de agente de gestor de colas local aparte. (Una conexión que no sea una conexión de *vía de acceso rápida* es una conexión *estándar* en este contexto. Para obtener más información, consulte “Conexión a un gestor de colas mediante la llamada MQCONNX” en la página 213.)

Cuando la aplicación emite `MQBEGIN`, tanto ella como el proceso de agente llaman a la función xa_open en la biblioteca de cliente de base de datos. En respuesta a esto, el código de la biblioteca de cliente de base de datos *se conecta* a la base de datos que va a participar en la unidad de trabajo *desde los procesos tanto de la aplicación como del gestor de recursos*. Estas conexiones de base de datos se mantienen mientras que la aplicación siga conectada al gestor de colas.

Esta es una consideración importante si la base de datos sólo tiene soporte para un número limitado de usuarios o conexiones, porque se establecen dos conexiones con la base de datos para dar soporte a un programa de aplicación.

Configuración de cliente/servidor

La biblioteca de cliente de base de datos que se carga en el gestor de colas y los procesos de aplicaciones de WebSphere MQ **deben** ser capaces de enviar a su servidor y recibir de él. Asegúrese de lo siguiente:

- Los archivos de configuración de cliente/servidor de la base de datos contienen los detalles correctos
- Las variables de entorno relevantes se establecen en el entorno del gestor de colas **y** de los procesos de aplicaciones

Creación de archivos de carga conmutada
WebSphere MQ se suministra con un archivo make de ejemplo, que se utiliza para crear archivos de carga conmutada para los gestores de bases de datos soportados.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

El archivo make de ejemplo, junto con todos los archivos de origen C asociados necesarios para construir los archivos de carga conmutada, se instala en los directorios siguientes:

- Para WebSphere MQ para Windows, en el directorio *MQ_INSTALLATION_PATH*\tools\c\samples\xatm\
- Para sistemas WebSphere MQ para UNIX and Linux , en el directorio *MQ_INSTALLATION_PATH*/samp/xatm/

Los módulos fuente de ejemplo utilizados para crear los archivos de carga conmutada son los siguientes:

- Para DB2, db2swit.c
- Para Oracle, oraswit.c
- Para Informix, infswit.c
- Para Sybase, sybswit.c

Al generar archivos de carga conmutada, instale archivos de carga conmutada de 32 bits en /var/mqm/exits e instale archivos de carga conmutada de 64 bits en /var/mqm/exits64.

Si tiene gestores de cola de 32 bits, el archivo make, xaswit.mak, instala un archivo de carga conmutada de 32 bits en /var/mqm/exits.

Si tiene gestores de cola de 64 bits, el archivo make, xaswit.mak, instala un archivo de carga conmutada de 32 bits en /var/mqm/exits y un archivo de carga conmutada de 64 bits en /var/mqm/exits64.

Seguridad de archivo

Es posible que el sistema operativo no pueda cargar el archivo de carga conmutada mediante WebSphere MQ, por motivos ajenos al control de WebSphere MQ. Si esto ocurre, los mensajes de error se escriben en los registros de errores de WebSphere MQ y, potencialmente, la llamada MQBEGIN puede fallar. Para asegurarse de que el sistema operativo no falla en la carga del archivo de carga conmutada, debe cumplir los requisitos siguientes:

1. El archivo de carga conmutada debe estar disponible en la ubicación dada en el archivo qm.ini.
2. El archivo de carga conmutada debe estar accesible para todos los procesos que necesiten cargarlo, incluyendo los procesos del gestor de colas y los procesos de la aplicación.
3. Todas las bibliotecas de las que dependa el archivo de carga conmutada, incluyendo las bibliotecas proporcionadas por el producto de la base de datos, deben estar presentes y accesibles.

Adición de información de configuración al gestor de colas
Cuando haya creado un archivo de carga conmutada para el gestor de bases de datos y lo haya colocado en una ubicación segura, deberá especificar dicha ubicación en el gestor de colas.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Para especificar la ubicación, realice los pasos siguientes:

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice WebSphere 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.
- En los demás sistemas, especifique los detalles del archivo de carga conmutada en la stanza XAResourceManager del archivo qm.ini del gestor de colas.

Añada una stanza XAResourceManager para la base de datos que el gestor de colas vaya a coordinar. El caso más normal es que haya sólo una base de datos y, por lo tanto, una sola stanza XAResourceManager. Para obtener detalles de configuraciones más complejas que impliquen el uso de varias bases de datos, consulte “Configuración de varias bases de datos” en la página 62. Los atributos de la stanza XAResourceManager son los siguientes:

Name=nombre
Serie de caracteres elegida por el usuario que identifica al gestor de recursos. En efecto, proporciona un nombre a la stanza XAResourceManager. Este nombre es obligatorio y puede tener hasta 31 caracteres de longitud.

El nombre elegido debe ser exclusivo; sólo debe haber una stanza XAResourceManager con este nombre en este archivo qm.ini. El nombre debe ser significativo ya que el gestor de colas lo utiliza para hacer referencia este gestor de recursos tanto en los mensajes como en la salida, cuando se utiliza el mandato `dspmqtrn`. (Consulte “Visualización de las unidades de trabajo pendientes con el mandato dspmqtrn” en la página 64 para obtener más información).

Cuando haya elegido un nombre y haya iniciado el gestor de colas, no cambie el atributo Name. Para obtener más detalles sobre cómo cambiar la información de configuración, consulte “Modificación de la información de configuración” en la página 67.

SwitchFile=nombre
Este es el nombre del archivo de carga conmutada XA que creó anteriormente. Este atributo es obligatorio. El código del gestor de colas y los procesos de aplicación de WebSphere MQ intentan cargar el archivo de carga conmutada en dos ocasiones:

1. Al iniciar el gestor de colas
2. Cuando realiza la primera llamada a MQBEGIN en el proceso de aplicación de WebSphere MQ

Los atributos de permiso y de seguridad del archivo de carga conmutada deben permitir que estos procesos realicen esta acción.

XAOpenString=serie
Esta es una serie de datos que el código de WebSphere MQ pasa en sus llamadas a la función xa_open del gestor de bases de datos. Es un atributo opcional; si se omite, se adopta una serie de caracteres de longitud cero.

El código del gestor de colas y los procesos de aplicación de WebSphere MQ llaman a la función xa_open en dos ocasiones:

1. Al iniciar el gestor de colas
2. Cuando realiza la primera llamada a MQBEGIN en el proceso de aplicación de WebSphere MQ

El formato de esta serie de caracteres es específico de cada base de datos, y se describirá en la documentación de dicho producto. En general, la serie de caracteres xa_open contiene información de autenticación (nombre de usuario y contraseña) para permitir realizar una conexión con la base de datos tanto en el gestor de colas como en los procesos de aplicación.

XACloseString=serie

Esta es una serie de datos que el código de WebSphere MQ pasa en sus llamadas a la función xa_close del gestor de bases de datos. Es un atributo opcional; si se omite, se adopta una serie de caracteres de longitud cero.

El código del gestor de colas y los procesos de aplicación de WebSphere MQ llaman a la función xa_close en dos ocasiones:

1. Al iniciar el gestor de colas
2. Cuando realiza una llamada a MQDISC en el proceso de aplicación de WebSphere MQ , después de haber realizado anteriormente una llamada a MQBEGIN

El formato de esta serie de caracteres es específico de cada base de datos, y se describirá en la documentación de dicho producto. En general, la serie está vacía, y es normal omitir el atributo XACloseString de la sección XAResourceManager.

ThreadOfControl=THREAD|PROCESS

El valor de ThreadOfControl puede ser THREAD o PROCESS. El gestor de colas lo utiliza con fines de serialización. Es un atributo opcional; si se omite, se adopta el valor PROCESS.

Si el código de cliente de base de datos permite que las hebras llamen a las funciones XA sin serialización, el valor para ThreadOfControl puede ser THREAD. El gestor de colas presupone que puede llamar a las funciones XA de la biblioteca compartida de cliente de la base de datos desde varias hebras al mismo tiempo, si es necesario.

Si el código de cliente de base de datos no permite que las hebras llamen a sus funciones XA de esta forma, el valor para ThreadOfControl debe ser PROCESS. En este caso, el gestor de colas serializa todas las llamadas a la biblioteca compartida de cliente de base de datos para que sólo se pueda realizar una llamada cada vez desde dentro de un proceso determinado. Es probable que también necesite asegurarse de que la aplicación realiza una serialización similar si se ejecuta en varias hebras.

Tenga en cuenta que este problema de que la base de datos pueda arreglársela con procesos de múltiples hebras de esta forma, es un problema del distribuidor de dicho producto. Consulte la documentación del producto de base de datos para obtener detalles sobre si puede establecer el atributo ThreadOfControl en THREAD o PROCESS. Es recomendable que, si puede, establezca ThreadOfControl en THREAD. Si duda, la opción *más segura* es establecer el atributo en PROCESS, aunque perderá las ventajas potenciales de rendimiento de que dispone al utilizar THREAD.

Escritura y modificación de las aplicaciones

Cómo implementar una unidad de trabajo global.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Los programas de aplicación de ejemplo para las unidades de trabajo globales del Escenario 1 que se proporcionan con una instalación de WebSphere MQ se describen en la publicación “Introducción a las unidades de trabajo” en la página 42.

En general, una unidad de trabajo global se implementa en una aplicación a través del siguiente método (en pseudocódigo):

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

El objetivo de MQBEGIN es indicar el principio de una unidad de trabajo global. El objetivo de MQCMIT es indicar el final de la unidad de trabajo global y completarla con todos los gestores de recursos participantes mediante un protocolo de confirmación en dos fases.

Entre MQBEGIN y MQCMIT, el gestor de colas no realiza ninguna llamada a la base de datos para actualizar sus recursos. Es decir, el único modo de cambiar las tablas de una base de datos es mediante su código (por ejemplo, SQL INSERT en el pseudocódigo).

La función del gestor de colas, mientras la base de datos esté implicada, es informarle sobre cuándo se ha iniciado una unidad de trabajo, cuándo ha concluido y si la unidad de trabajo global debe confirmarse o restituirse.

Siempre que su aplicación esté involucrada, el gestor de colas realizará dos funciones: un gestor de recursos (en el que los recursos son mensajes en colas) y el gestor de transacciones para la unidad de trabajo global.

Empiece con los programas de ejemplo proporcionados y trabaje a través de las diversas llamadas de API de base de datos y WebSphere MQ que se están realizando en dichos programas. Las llamadas de API afectadas están totalmente documentadas en "Programas WebSphere MQ de ejemplo" en la página 98, Tipos de datos utilizados en la MQIy (en el caso de la propia API de la base de datos) la propia documentación de la base de datos.

Pruebas del sistema
Sólo se puede saber si tanto la aplicación como el sistema están correctamente configurados, ejecutándolos durante la prueba. Puede probar la configuración del sistema (comunicación correcta entre el gestor de colas y la base de datos) creando y ejecutando uno de los programas de ejemplo suministrados.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Configuración de Db2
Información de soporte y configuración de DB2 .

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Los niveles soportados de Db2 se definen en la página IBM WebSphere MQ requisitos detallados del sistema .

Nota: Las instancias de 32 bits de Db2 no están soportadas en plataformas donde el gestor de colas es de 64 bits.

Realice lo siguiente:

1. Comprobar los valores de las variables de entorno.
2. Cree el archivo de carga conmutada Db2 .
3. Añadir información de configuración del gestor de recursos.
4. Cambie los parámetros de configuración de Db2 si es necesario.

Lea esta información junto con la información general que se facilita en "Configuración del sistema para la coordinación de bases de datos" en la página 48.

Aviso: Si ejecuta `db2profile` en las plataformas UNIX and Linux, se establecen las variables de entorno LIBPATH y LD_LIBRARY_PATH. Es aconsejable `unset` estas variables de entorno; consulte la publicación *Guía de iniciación rápida* adecuada.

Comprobación de los valores de las variables de entorno de Db2

Asegúrese de que las variables de entorno de Db2 estén establecidas para los procesos del gestor de colas ***, así como en*** los procesos de aplicación. En concreto, debe establecer siempre la variable de entorno DB2INSTANCE ***antes*** de iniciar el gestor de colas. La variable de entorno DB2INSTANCE identifica la instancia de Db2 que contiene las bases de datos Db2 que se están actualizando. Por ejemplo:

- En sistemas UNIX and Linux , utilice:

```
export DB2INSTANCE=db2inst1
```

- En sistemas Windows , utilice:

```
set DB2INSTANCE=DB2
```

En Windows con una base de datos Db2 , debe añadir el usuario MUSR_MQADMIN al grupo DB2USERS , para habilitar el inicio del gestor de colas.

Creación del archivo de carga conmutada Db2

La forma más fácil de crear el archivo de carga conmutada Db2 es utilizar el archivo de ejemplo xaswit.mak, que WebSphere MQ proporciona para crear los archivos de carga conmutada para una variedad de productos de base de datos.

En sistemas Windows , puede encontrar xaswit.mak en el directorio *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ . Para crear el archivo de carga conmutada Db2 con Microsoft Visual C++, utilice:

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

El archivo de conmutación generado se coloca en `c:\Program Files\IBM\WebSphere MQ\exits`.

Puede encontrar xaswit.mak en el directorio *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ .

Edite xaswit.mak para *descomentar* las líneas adecuadas a la versión de Db2 que está utilizando. A continuación, ejecute el archivo make utilizando el mandato:

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

El archivo de carga conmutada de 32 bits generado se coloca en `/var/mqm/exits`.

El archivo de carga conmutada de 64 bits generado se coloca en `/var/mqm/exits64`.

Adición de información de configuración del gestor de recursos para Db2

Debe modificar la información de configuración para que el gestor de colas declare Db2 como participante en unidades de trabajo globales. En “Adición de información de configuración al gestor de colas” en la página 50 se describe, de forma más detallada, cómo modificar la información de configuración de este modo.

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice WebSphere 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.
- En los demás sistemas, especifique los detalles del archivo de carga conmutada en la stanza XAResourceManager del archivo qm.ini del gestor de colas.

Figura 9 en la página 54 es un ejemplo de UNIX , que muestra una entrada XAResourceManager donde la base de datos que se va a coordinar se denomina `mydbname`, este nombre se especifica en `XAOpenString`:

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

Figura 9. Entrada XAResourceManager de ejemplo para Db2 en plataformas UNIX

Nota:

1. `ThreadOfControl=THREAD` no se puede utilizar con versiones de Db2 anteriores a la versión 8. Establezca `ThreadOfControl` y el parámetro XAOpenString `toc` en una de las siguientes combinaciones:
   - `ThreadOfControl=THREAD` y `toc=t`
   - `ThreadOfControl=PROCESS` y `toc=p`

   Si utiliza el archivo de carga conmutada XA jdbcdb2 para habilitar la coordinación JDBC/JTA, debe utilizar `ThreadOfControl=PROCESS` y `toc=p`.

Cambio de los parámetros de configuración de Db2

Para cada base de datos Db2 que el gestor de colas está coordinando, debe establecer privilegios de base de datos, cambiar el parámetro tp_mon_name y restablecer el parámetro maxappls. Para hacerlo, efectúe los pasos siguientes:

Establecer los privilegios de la base de datos

Los procesos del gestor de colas se ejecutan con el usuario efectivo y el grupo mqm en sistemas UNIX and Linux. En sistemas Windows , se ejecutan como el usuario que ha iniciado el gestor de colas. Este puede ser:

1. El usuario que ha emitido el mandato `strmqm`, o bien
2. El usuario bajo el que se ejecuta el servidor IBM MQSeries Service COM

De forma predeterminada, este usuario se denomina MUSR_MQADMIN.

Si no ha especificado un nombre de usuario y una contraseña en la serie xa_open, **el usuario con el que se ejecuta el gestor de colas** lo utiliza Db2 para autenticar la llamada xa_open. Si este usuario (por ejemplo, el usuario mqm en sistemas UNIX and Linux) no tiene privilegios mínimos en la base de datos, la base de datos se niega a autenticar la llamada xa_open.

En el proceso de aplicaciones se le aplican las mismas consideraciones. Si no ha especificado un nombre de usuario y una contraseña en la serie xa_open, Db2 utiliza el usuario bajo el que se ejecuta la aplicación para autenticar la llamada xa_open que se realiza durante el primer MQBEGIN. Este usuario también debe tener privilegios mínimos en la base de datos para que esto funcione.

Por ejemplo, otorgue al usuario mqm autorización de conexión en la base de datos mydbname emitiendo los siguientes mandatos Db2 :

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

Consulte “Consideraciones acerca de la seguridad” en la página 63 para obtener más información sobre seguridad.

Windows **Cambiar el parámetro TP_MON_NAME**

Para Db2 solo para sistemas Windows , cambie el parámetro de configuración TP_MON_NAME para denominar la DLL que Db2 utiliza para llamar al gestor de colas para el registro dinámico.

Utilice el mandato `db2 update dbm cfg using TP_MON_NAME mqmax` para denominar MQMAX.DLL como la biblioteca que Db2 utiliza para llamar al gestor de colas. Debe encontrarse en un directorio de PATH.

Restaurar el parámetro maxappls
Es posible que tenga que revisar el valor del parámetro *maxappls*, que limita el número máximo de aplicaciones que pueden estar conectadas a una base de datos. Consulte el apartado "Instalación y configuración del producto de base de datos" en la página 48.

Configuración de Oracle

Información de soporte y configuración de Oracle.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Complete los pasos siguientes:

1. Comprobar los valores de las variables de entorno.
2. Crear el archivo de carga conmutada de Oracle.
3. Añadir información de configuración del gestor de recursos.
4. Cambiar los parámetros de configuración de Oracle si es necesario.

La lista actual de niveles de Oracle soportados por IBM WebSphere MQ se proporciona en la página IBM WebSphere MQ requisitos detallados del sistema.

Comprobación de los valores de las variables de entorno de Oracle

Asegúrese de que las variables de entorno de Oracle están configuradas para los procesos del gestor de colas así como para los procesos de aplicaciones. Sobre todo, establezca siempre las variables de entorno siguientes antes de iniciar el gestor de colas:

ORACLE_HOME
El directorio inicial de Oracle. Por ejemplo, en sistemas UNIX and Linux , utilice:

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

En sistemas Windows, utilice:

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

ORACLE_SID
El SID de Oracle que se va a utilizar. Si utiliza Net8 para conectividad cliente/servidor, es posible que no necesite configurar esta variable de entorno. Consulte la documentación de Oracle.

A continuación figura un ejemplo de cómo establecer esta variable de entorno, en los sistemas UNIX and Linux:

```
export ORACLE_SID=sid1
```

El equivalente en sistemas Windows es:

```
set ORACLE_SID=sid1
```

Nota: La variable de entorno PATH debe establecerse de modo que incluya el directorio de archivos binarios (por ejemplo, `ORACLE_INSTALL_DIR/VERSION/32BIT_NAME/bin` o `ORACLE_INSTALL_DIR/VERSION/64BIT_NAME/bin`), de lo contrario podría ver un mensaje indicando que faltan las bibliotecas oraclient en la máquina.

Si ejecuta gestores de colas en sistemas Windows de 64 bits, se deben instalar los clientes Oracle de 64 bits y 32 bits. Debe instalar ambos clientes porque el gestor de colas se ejecuta como procesos de 32 bits que utilizan un archivo de carga conmutada de 32 bits, que a su vez debe iniciar una dll de cliente Oracle de 32 bits.

El archivo de carga conmutada, que cargan los gestores de colas de 64 bits, debe acceder a las bibliotecas de cliente 64 de bits Oracle. Los gestores de colas de 32 bits deben acceder al cliente Oracle de 32 bits cuando IBM WebSphere MQ se ejecuta en un sistema Windows de 64 bits.

Creación del archivo de carga conmutada de Oracle.

Para crear el archivo de carga conmutada de Oracle, utilice el archivo de ejemplo `xaswit.mak`, que IBM WebSphere MQ proporciona para crear los archivos de carga conmutada para varios productos de base de datos. En sistemas Windows , puede encontrar `xaswit.mak` en el directorio `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. Para crear el archivo de carga conmutada Oracle con Microsoft Visual C++, utilice: `nmake /f xaswit.mak oraswit.dll`

El archivo de conmutación generado se coloca en *MQ_INSTALLATION_PATH*`\exits`.*MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM WebSphere MQ .

Puede encontrar `xaswit.mak` en el directorio *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM WebSphere MQ .

Edite el archivo `xaswit.mak` para eliminar los comentarios de las líneas correspondientes a la versión de Oracle que está utilizando. A continuación, ejecute el archivo make utilizando el mandato:

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

El archivo de carga conmutada de 32 bits generado se coloca en `/var/mqm/exits`.

El archivo de carga conmutada de 64 bits generado se coloca en `/var/mqm/exits64`.

Adición de información de configuración del gestor de recursos para Oracle

Debe modificar la información de configuración del gestor de colas para declarar Oracle como participante en unidades de trabajo globales. En “Adición de información de configuración al gestor de colas” en la página 50 se describe, de forma más detallada, cómo modificar de esta manera la información de configuración del gestor de colas.

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice IBM WebSphere 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.
- En todos los demás sistemas, especifique los detalles del archivo de carga conmutada en la stanza XAResourceManager del archivo `qm.ini` del gestor de colas.

La Figura 10 en la página 56 es un ejemplo de sistemas UNIX and Linux en la que se muestra una entrada XAResourceManager. Debe añadir un valor `LogDir` a la serie de apertura de XA de forma que toda la información sobre errores y rastreo pueda anotarse en el mismo sitio.

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

Figura 10. Ejemplo de entrada XAResourceManager para Oracle en plataformas UNIX and Linux

Nota:

1. En la Figura 10 en la página 56, la serie de caracteres xa_open se ha utilizado con cuatro parámetros. Se pueden incluir parámetros adicionales tal como se describe en la documentación de Oracle.
2. Al utilizar el parámetro IBM WebSphere MQ `ThreadOfControl=THREAD` , debe utilizar el parámetro Oracle `+threads=true` en la stanza XAResourceManager .

Consulte la publicación *Oracle8 Server Application Developer's Guide* para obtener más información sobre la serie xa_open.

Modificación de parámetros de configuración de Oracle

Para cada base de datos de Oracle que el gestor de colas esté coordinando, deberá revisar el máximo de sesiones y establecer privilegios de base de datos. Para ello, realice estos pasos:

Revisar el máximo de sesiones
Es posible que tenga que revisar los valores de LICENSE_MAX_SESSIONS y PROCESSES a fin de tener en cuenta las conexiones adicionales que requieran los procesos pertenecientes al gestor de colas. Consulte “Instalación y configuración del producto de base de datos” en la página 48 para obtener más detalles.

Establecer los privilegios de la base de datos
El nombre de usuario de Oracle especificado en la serie xa_open debe tener privilegios para acceder a la vista DBA_PENDING_TRANSACTIONS, tal y como se describe en la documentación de Oracle.

Puede otorgar el privilegio necesario utilizando el siguiente mandato de ejemplo:

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

Configuración de Informix

Información de soporte y configuración de Informix .

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Complete los pasos siguientes:

1. Asegúrese de que ha instalado el SDK de cliente de Informix adecuado:
   - Los gestores de colas y las aplicaciones de 32 bits requieren un SDK de cliente de Informix de 32 bits.
   - Los gestores de colas y las aplicaciones de 64 bits requieren un SDK de cliente de Informix de 64 bits.
2. Asegúrese de que las bases de datos Informix se hayan creado correctamente .
3. Comprobar los valores de las variables de entorno.
4. Cree el archivo de carga conmutada Informix .
5. Añadir información de configuración del gestor de recursos.

Se proporciona una lista actual de niveles de Informix soportados por WebSphere MQ en la páginaIBM WebSphere MQ requisitos detallados del sistema .

Asegurarse de que las bases de datos Informix se han creado correctamente

Cada base de datos Informix que debe coordinar un gestor de colas WebSphere MQ debe crearse especificando el parámetro `log` . Por ejemplo:

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

Los gestores de colas de WebSphere MQ no pueden coordinar las bases de datos Informix que no tienen el parámetro `log` especificado en la creación. Si un gestor de colas intenta coordinar una base de datos

Informix que no tiene el parámetro `log` especificado en la creación, la llamada xa_open a Informix falla y se generan varios errores FFST .

Comprobación de los valores de las variables de entorno de Informix

Asegúrese de que las variables de entorno de Informix estén establecidas para los procesos del gestor de colas ***así como en*** los procesos de aplicación. Sobre todo, establezca siempre las variables de entorno siguientes **antes** de iniciar el gestor de colas:

INFORMIXDIR
El directorio de la instalación del producto Informix .

- Para aplicaciones de 32 bits UNIX and Linux, utilice el mandato siguiente:

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

- Para aplicaciones de 64 bits UNIX and Linux, utilice el mandato siguiente:

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

- Para aplicaciones Windows , utilice el mandato siguiente:

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

Para sistemas que tienen gestores de colas de 64 bits que deben dar soporte a aplicaciones de 32 bits y de 64 bits, necesita instalados los SDK de cliente de Informix de 32 bits y 64 bits. El archivo make de ejemplo, `xaswit.mak`, que se utiliza para crear un archivo de carga conmutada, también establece los dos directorios de instalación del producto.

INFORMIXSERVER
El nombre del servidor Informix . Por ejemplo, en sistemas UNIX and Linux , utilice:

```
export INFORMIXSERVER=hostname_1
```

En sistemas Windows , utilice:

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG
El nombre del archivo de configuración del servidor Informix . Por ejemplo, en sistemas UNIX and Linux , utilice:

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

En sistemas Windows , utilice:

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

Creación del archivo de carga conmutada Informix

Para crear el archivo de carga conmutada Informix , utilice el archivo de ejemplo xaswit.mak, que proporciona WebSphere MQ para crear los archivos de carga conmutada para diversos productos de base de datos. En sistemas Windows , puede encontrar xaswit.mak en el directorio *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm`. *MQ_INSTALLATION_PATH* representa

el directorio de alto nivel en el que está instalado WebSphere MQ . Para crear el archivo de carga conmutada Informix con Microsoft Visual C++, utilice:

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

El archivo de conmutación generado se coloca en `c:\Program Files\IBM\WebSphere MQ\exits`.

Puede encontrar xaswit.mak en el directorio *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ .

Edite xaswit.mak para *descomentar* las líneas adecuadas a la versión de Informix que está utilizando. A continuación, ejecute el archivo make utilizando el mandato:

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

El archivo de carga conmutada de 32 bits generado se coloca en `/var/mqm/exits`.

El archivo de carga conmutada de 64 bits generado se coloca en `/var/mqm/exits64`.

Adición de información de configuración del gestor de recursos para Informix

Debe modificar la información de configuración para que el gestor de colas declare Informix como participante en unidades de trabajo globales. En "Adición de información de configuración al gestor de colas" en la página 50 se describe, de forma más detallada, cómo modificar de esta manera la información de configuración del gestor de colas.

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice WebSphere 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.
- En todos los demás sistemas, especifique los detalles del archivo de carga conmutada en la stanza XAResourceManager del archivo qm.ini del gestor de colas.

Figura 11 en la página 59 es un ejemplo de UNIX , que muestra una entrada qm.ini XAResourceManager donde la base de datos que se va a coordinar se denomina `mydbname`, este nombre se especifica en `XAOpenString`:

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

Figura 11. Entrada XAResourceManager de ejemplo para Informix en plataformas UNIX

Nota: De forma predeterminada, el ejemplo xaswit.mak en plataformas UNIX crea un archivo de carga conmutada que utiliza bibliotecas Informix con hebras. Debe asegurarse de que ThreadOfControl esté establecido en THREAD cuando utilice estas bibliotecas de Informix . En Figura 11 en la página 59, el atributo ThreadOfControl de la stanza XAResourceManager del archivo qm.ini está establecida en THREAD. Cuando se especifica THREAD, las aplicaciones deben compilarse utilizando las bibliotecas Informix con hebras y las bibliotecas de API con hebras de WebSphere MQ .

El atributo XAOpenString debe contener el nombre de base de datos, seguido del símbolo @ y, a continuación, seguido del nombre de servidor Informix .

Para utilizar las bibliotecas Informix sin hebras, debe asegurarse de que el archivo qm.ini XAResourceManager atributo de stanza ThreadOfControl está establecido en PROCESS. También debe realizar los cambios siguientes en el archivo xaswit.mak de ejemplo:

1. Elimine el comentario de la generación de un archivo de carga conmutada sin hebras.
2. Convierta en comentario la generación del archivo de carga conmutada con hebras.

Configuración de Sybase

Información de soporte y configuración de Sybase.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Complete los pasos siguientes:

1. Asegúrese de que ha instalado las bibliotecas XA de Sybase, por ejemplo, instalando la opción DTM para XA.
2. Comprobar los valores de las variables de entorno.
3. Habilitar el soporte de XA de Sybase.
4. Crear el archivo de carga conmutada de Sybase.
5. Añadir información de configuración del gestor de recursos.

Se proporciona una lista actual de niveles de Sybase soportados por WebSphere MQ en la página IBM WebSphere MQ requisitos detallados del sistema .

Comprobación de los valores de las variables de entorno de Sybase

Asegúrese de que las variables de entorno de Sybase están configuradas para los procesos del gestor de colas ***así como*** para los procesos de aplicaciones. Sobre todo, establezca siempre las variables de entorno siguientes **antes** de iniciar el gestor de colas:

SYBASE
La ubicación de la instalación del producto Sybase. Por ejemplo, en sistemas UNIX and Linux , utilice:

```
export SYBASE=/sybase
```

En sistemas Windows , utilice:

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

SYBASE_OCS
El directorio de SYBASE en el que haya instalado los archivos de cliente de Sybase. Por ejemplo, en sistemas UNIX and Linux , utilice:

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

En sistemas Windows , utilice:

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

Habilitación del soporte de XA de Sybase

Dentro del archivo de configuración XA de Sybase $SYBASE/$SYBASE_OCS/`xa_config`, defina un gestor de recursos lógicos (LRM) para cada conexión con el servidor Sybase que se está actualizando. En la Figura 12 en la página 61se muestra un ejemplo del contenido de $SYBASE/$SYBASE_OCS/`xa_config` .

```
# The first line must always be a comment

[xa]

  LRM=lrmname
  server=servername
```

Figura 12. Ejemplo del contenido de $SYBASE/$SYBASE_OCS/xa_config

Creación del archivo de carga conmutada de Sybase

Para crear el archivo de carga conmutada Sybase , utilice los archivos de ejemplo proporcionados con WebSphere MQ.En sistemas Windows , puede encontrar xaswit.mak en el directorio `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. Para crear el archivo de carga conmutada Sybase con Microsoft Visual C++, utilice:

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

El archivo de conmutación generado se coloca en `c:\Program Files\IBM\WebSphere MQ\exits`.

Puede encontrar xaswit.mak en el directorio *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ .

Edite el archivo xaswit.mak para *descomentar* las líneas correspondientes a la versión de Sybase que está utilizando. A continuación, ejecute el archivo make utilizando el mandato:

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

El archivo de carga conmutada de 32 bits generado se coloca en `/var/mqm/exits`.

El archivo de carga conmutada de 64 bits generado se coloca en `/var/mqm/exits64`.

Adición de información de configuración del gestor de recursos para Sybase

Debe modificar la información de configuración del gestor de colas para declarar Sybase como participante en unidades de trabajo globales. En “Adición de información de configuración al gestor de colas” en la página 50 se describe, de forma más detallada, cómo modificar la información de configuración del gestor de colas.

- En sistemas Windows y Linux (plataformasx86 y x86-64), utilice WebSphere 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.
- En los demás sistemas, especifique los detalles del archivo de carga conmutada en la stanza XAResourceManager del archivo qm.ini del gestor de colas.

En la Figura 13 en la página 62 se muestra un ejemplo para UNIX and Linux, que utiliza la base de datos asociada a la definición LRM *nombre_lrm* en el archivo de configuración XA de Sybase, $SYBASE/$SYBASE_OCS/`xa_config`. Si desea anotar llamadas de función XA, debe incluir un nombre de archivo de anotaciones:

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

Figura 13. Ejemplo de entrada XAResourceManager para Sybase en plataformas UNIX and Linux

Utilización de programas multihebra con Sybase

Si utiliza programas multihebra con unidades de trabajo globales de WebSphere MQ que incorporan actualizaciones a Sybase, ***debe*** utilizar el valor THREAD para el parámetro de control ThreadOf. Asegúrese también de enlazar el programa (y el archivo de carga conmutada) a las bibliotecas Sybase preparadas para hebras (las versiones _r). En Figura 13 en la página 62 se muestra La utilización del valor THREAD para el parámetro ThreadOfControl.

Configuración de varias bases de datos

Si desea configurar el gestor de colas para poder incluir actualizaciones a varias bases de datos dentro de unidades de trabajo globales, añada una sección XAResourceManager para cada base de datos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Si las bases de datos las gestiona un mismo gestor de bases de datos, cada sección define una base de datos independiente. Cada sección especifica el mismo archivo de conmutación (*SwitchFile*), pero el contenido de cada serie *XAOpenString* es distinto porque especifica el nombre de la base de datos que está actualizándose. Por ejemplo, las stanzas que se muestran en Figura 14 en la página 62 configuran el gestor de colas con las bases de datos Db2 *MQBankDB* y *MQFeeDB* en sistemas UNIX and Linux .

Importante: No puede tener varias secciones que apunten a la misma base de datos. Esta configuración no funciona en ningún caso y si intenta esta configuración fallará.

Recibirá errores con el formato `when the MQ code makes its second xa_open call in any process in this environment, the database software fails the second xa_open with a -5 error, XAER_INVAL`.

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

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

Figura 14. Entradas de ejemplo de XAResourceManager para varias bases de datos Db2

Si las bases de datos que se van a actualizar las gestionan distintos gestores de bases de datos, añada una stanza XAResourceManager para cada una de ellas. En este caso, cada stanza especifica un archivo de conmutación (*SwitchFile*) diferente. Por ejemplo, si *MQFeeDB* está gestionado por Oracle en lugar de DB2, utilice las siguientes stanzas en sistemas UNIX and Linux :

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

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

Figura 15. Entradas XAResourceManager de ejemplo para una base de datos DB2 y Oracle

En principio, no hay ningún límite en el número de instancias de base de datos que pueden configurarse con un solo gestor de colas.

Nota: Para obtener información sobre el soporte para incluir bases de datos Informix en varias actualizaciones de base de datos dentro de unidades de trabajo globales, consulte el archivo léame del producto.

Consideraciones acerca de la seguridad

Consideraciones sobre la ejecución de la base de datos bajo el modelo XA.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

La siguiente información se proporciona únicamente como guía. En todos los casos, consulte la documentación proporcionada con el gestor de colas a fin de determinar las implicaciones de seguridad que tiene ejecutar la base de datos bajo el modelo XA.

Un proceso de aplicaciones indica el inicio de una unidad de trabajo global mediante la utilización del verbo MQBEGIN. La primera llamada MQBEGIN emitida por una aplicación se conecta con todas las bases de datos participantes llamando a su código de biblioteca de cliente en el punto de entrada xa_open. Todos los gestores de bases de datos proporcionan un mecanismo para facilitar un ID de usuario y una contraseña en sus XAOpenString. Se trata del único momento en que la información de autenticación fluye.

Tenga en cuenta que, en las plataformas UNIX and Linux, las aplicaciones de vía de acceso rápida deben ejecutarse con un ID de usuario efectivo de mqm al realizar llamadas MQI.

Consideraciones cuando se pierde el contacto con el gestor de recursos XA

El gestor de colas tolera que los gestores de bases de datos no estén disponibles. Esto significa que se puede iniciar y detener el gestor de colas independientemente del servidor de bases de datos. Cuando se restaura el contacto, el gestor de colas y la base de datos se resincronizan. También puede utilizar el mandato rsvmqtrn para resolver manualmente unidades de trabajo pendientes.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

En operaciones normales, una vez realizados los pasos de configuración, la administración será mínima. El trabajo de administración es más fácil debido a que el gestor de colas tolera que los gestores de bases de datos no estén disponibles. En particular, esto significa que:

- El gestor de colas puede iniciarse en cualquier momento sin necesidad de iniciar primero cada uno de los gestores de bases de datos.
- No es necesario detener y reiniciar el gestor de colas si alguno de los gestores de bases de datos deja de estar disponible.

Esto permite iniciar y detener el gestor de colas independientemente del servidor de bases de datos.

Siempre que se pierde el contacto entre el gestor de colas y una base de datos, ambos deberán volver a sincronizarse cuando vuelvan a estar disponibles. La resincronización es el proceso mediante el cual se completan todas las unidades de trabajo pendientes referentes a una base de datos. En general, esto ocurre de forma automática, sin intervención del usuario. El gestor de colas solicita a la base de datos una lista de las unidades de trabajo en las que tiene operaciones pendientes. A continuación, indica a la base de datos que confirme o restituya cada una de las unidades de trabajo pendientes.

Cuando se inicia un gestor de colas, éste se vuelve a sincronizar con cada una de las bases de datos. Cuando una base de datos individual deja de estar disponible, únicamente dicha base de datos deberá resincronizarse la próxima vez que el gestor de colas advierta que vuelve a estar disponible.

A medida que se inician nuevas unidades de trabajo con MQBEGIN, el gestor de colas recupera automáticamente el contacto con la base de datos no disponible anteriormente. Esto lo lleva a cabo llamando a la función xa_open de la biblioteca de cliente de la base de datos. Si esta llamada de xa_open no se ejecuta correctamente, MQBEGIN vuelve con un código de terminación de MQCC_WARNING y un código de razón de MQRC_PARTICIPANT_NOT_AVAILABLE. Puede volver a intentar la ejecución de MQBEGIN más tarde.

No siga intentando realizar una unidad de trabajo global que incluya actualizaciones a una base de datos que hay indicado error al llevar a cabo MQBEGIN. No habrá ninguna conexión con esa base de datos a través de la que se puedan realizar actualizaciones. Las únicas opciones son terminar el programa o volver a intentar la ejecución de MQBEGIN periódicamente con la esperanza de que la base de datos vuelva a estar de nuevo disponible.

Alternativamente, puede utilizar el mandato `rsvmqtrn` para resolver explícitamente todas las unidades de trabajo pendientes.

Unidades de trabajo pendientes

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Una base de datos puede quedar con unidades de trabajo dudosas si se pierde el contacto con el gestor de colas después de que se haya indicado al gestor de bases de datos que se prepare. Hasta que el servidor de bases de datos reciba el resultado del gestor de colas (confirmación o restitución), deberá retener los bloqueos de base de datos asociados a las actualizaciones.

Como los bloqueos impiden que otras aplicaciones actualicen o lean los registros de las bases de datos, la resincronización debe realizarse lo antes posible.

Si, por algún motivo, no puede esperar a que el gestor de colas se resincronice automáticamente con la base de datos, puede utilizar los recursos que proporciona el gestor de bases de datos para confirmar o restituir manualmente las actualizaciones de la base de datos. En el documento *Proceso de transacciones distribuidas X/Open: la especificación XA*, esto se denomina tomar una decisión *heurística*. Sólo debe utilizarse como último recurso puesto que existe la posibilidad de comprometer la integridad de los datos; por ejemplo, puede restituir erróneamente las actualizaciones de base de datos cuando todos los demás participantes hayan confirmado sus actualizaciones.

Es mucho mejor reiniciar el gestor de colas, o utilizar el mandato `rsvmqtrn` cuando se haya reiniciado la base de datos, para iniciar la resincronización automática.

Visualización de las unidades de trabajo pendientes con el mandato dspmqtrn

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Mientras un gestor de bases de datos no está disponible, se puede utilizar el mandato **dspmqtrn** para comprobar el estado de las unidades de trabajo globales pendientes referentes a dicha base de datos.

El mandato **dspmqtrn** sólo muestra las unidades de trabajo en las que uno o más participantes están en duda. Los participantes están esperando la decisión del gestor de colas para confirmar o retrotraer las actualizaciones preparadas.

Para cada una de estas unidades de trabajo globales, el estado de cada participante se visualiza en la salida de **dspmqtrn**. Si la unidad de trabajo no ha actualizado los recursos de un gestor de recursos determinado, no se visualiza.

Con respecto a una unidad de trabajo pendiente, se dice que un gestor de recursos ha realizado una de las siguientes funciones:

Preparado
El gestor de recursos está preparado para confirmar sus actualizaciones.

Confirmado
El gestor de recursos ha confirmado sus actualizaciones.

Retrotraídas
El gestor de recursos ha retrotraído sus actualizaciones.

Participante
El gestor de recursos es un participante, pero no ha preparado, confirmado ni restituido sus actualizaciones.

Cuando se reinicia el gestor de colas, éste solicita a cada base de datos que tenga una stanza XAResourceManager una lista de sus unidades de trabajo globales pendientes. Si la base de datos no se ha reiniciado, o no está disponible por otras razones, el gestor de colas no podrá entregarle todavía los resultados finales de dichas unidades de trabajo. El resultado de las unidades de trabajo pendientes se entrega a la base de datos cuando se presente la primera oportunidad de disponibilidad de la base de datos.

En este caso, se notifica que el gestor de bases de datos está en estado *preparado* hasta que se produzca la resincronización.

Siempre que el mandato `dspmqtrn` muestre una unidad de trabajo en estado dudoso, en primer lugar muestra todos los gestores de recursos posibles que podrían estar participando. A estos se les asigna un identificador exclusivo, *RMId*, que se utiliza en lugar del *Nombre* de los gestores de recursos al informar de su estado con respecto a una unidad de trabajo dudosa.

El ejemplo de salida de dspmqtrn muestra el resultado de la emisión del mandato siguiente:

```
dspmqtrn -m MY_QMGR
```

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

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

donde *Número de transacción* es el ID de la transacción que se puede utilizar con el mandato rsvmqtrn. Consulte AMQ7000-7999: Producto WebSphere MQ para obtener más información sobre el mensaje AMQ7056 . Las variables *XID* forman parte de la *Especificación X/Open XA*; para obtener la información más actualizada sobre esta especificación, consulte: https://publications.opengroup.org/c193.

Figura 16. Ejemplo de salida de dspmqtrn

La salida del Ejemplo de salida de dspmqtrn muestra que existen tres gestores de recursos asociados al gestor de colas. El primero es el gestor de recursos `0`, que es el gestor de colas propiamente dicho. Las otras dos instancias del gestor de recursos son las bases de datos `MQBankDB` y `MQFeeDB` Db2 .

El ejemplo sólo muestra una sola unidad de trabajo pendiente. Se emite un mensaje para los tres gestores de recursos, lo que significa que se han realizado actualizaciones en el gestor de colas y en ambas bases de datos Db2 dentro de la unidad de trabajo.

Las actualizaciones realizadas en el gestor de colas, el gestor de recursos **0**, han sido *comprometido*. Las actualizaciones de las bases de datos Db2 están en estado *preparado* , lo que significa que Db2 debe haber dejado de estar disponible antes de que se le llamara para confirmar las actualizaciones en las bases de datos *MQBankDB* y *MQFeeDB* .

La unidad de trabajo dudosa tiene un identificador externo denominado XID (*id de transacción*). Se trata de un fragmento de datos proporcionado a Db2 por el gestor de colas para identificar su parte de la unidad de trabajo global.

Resolución de las unidades de trabajo pendientes con el mandato rsvmqtrn
Las unidades de trabajo pendientes se completan cuando el gestor de colas y DB2 se resincronizan.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

La salida que se muestra en la Figura 16 en la página 65 muestra una única unidad de trabajo pendiente en la que la decisión de confirmación todavía no se ha entregado a ambas bases de datos DB2 .

Para completar esta unidad de trabajo, el gestor de colas y DB2 deben resincronizarse cuando DB2 pase a estar disponible. El gestor de colas utiliza el inicio de nuevas unidades de trabajo como una oportunidad para recuperar el contacto con DB2. Como alternativa, puede indicar al gestor de colas que se resincronice explícitamente mediante el mandato **rsvmqtrn**.

Hágalo poco después de que se haya reiniciado DB2 , para que los bloqueos de base de datos asociados a la unidad de trabajo pendiente se liberen lo más rápidamente posible. Utilice la opción -a, que indica al gestor de colas que resuelva todas las unidades de trabajo pendientes. En el ejemplo siguiente, DB2 se ha reiniciado, por lo que el gestor de colas puede resolver la unidad de trabajo pendiente:

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

Resultados mixtos y errores
Aunque el gestor de colas utiliza el protocolo de confirmación en dos fases, esto no elimina completamente la posibilidad de que algunas unidades de trabajo terminen con resultados mixtos. Los resultados mixtos se producen cuando algunos participantes confirman sus actualizaciones mientras que otros las restituyen.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Las unidades de trabajo que terminan con un resultado mixto tienen graves repercusiones debido a que el estado de los recursos compartidos que se hayan actualizado como una sola unidad de trabajo deja de ser coherente.

Los resultados mixtos se producen principalmente cuando se toman decisiones heurísticas sobre unidades de trabajo, en vez de permitir que el gestor de colas resuelva por sí mismo las unidades de trabajo pendientes. Dichas decisiones están fuera del control del gestor de colas.

Siempre que el gestor de colas detecta un resultado mixto, genera información FFST y documenta la anomalía en sus registros de errores, con uno de dos mensajes:

- Si un gestor de bases de datos ha efectuado una restitución en lugar de una confirmación:

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

- Si un gestor de bases de datos ha efectuado una confirmación en vez de una restitución:

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

Se emiten otros mensajes para identificar las bases de datos dañadas heurísticamente. Es responsabilidad del usuario restaurar localmente la coherencia en las bases de datos afectadas. Este es un procedimiento complicado en el que primero es necesario determinar la actualización que se ha confirmado o restituido erróneamente y, después, debe deshacerse o rehacerse manualmente el cambio en la base de datos.

Modificación de la información de configuración
Cuando el gestor de colas haya empezado a coordinar correctamente unidades de trabajo globales, no modifique la información de configuración del gestor de recursos.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Si necesita modificar la información de configuración puede hacerlo siempre que lo desee, pero los cambios no surtirán efecto hasta que reinicie el gestor de colas.

Si elimina la información de configuración del gestor de recursos para una base de datos, de hecho estará eliminando la posibilidad de que el gestor se comunique con dicho gestor de colas.

Nunca cambie el atributo *Name* en la información de configuración del gestor de recursos. Este atributo identifica de forma exclusiva la instancia del gestor de bases de datos ante el gestor de colas. Si cambia este identificador exclusivo, el gestor de colas presupone que la base de datos se ha eliminado y que se ha añadido una instancia totalmente nueva. El gestor de colas sigue asociando las unidades de trabajo pendientes con el *nombre* antiguo y es posible que deje la base de datos en estado pendiente.

Eliminación de instancias de gestor de bases de datos
Si necesita eliminar permanentemente una base de datos de la configuración, asegúrese de que la base de datos no está pendiente antes de reiniciar el gestor de colas.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Las bases de datos proporcionan mandatos para listar las transacciones pendientes. Si hay alguna transacción pendiente, permita primero que el gestor de colas se resincronice con el gestor de bases de datos. Para ello, inicie el gestor de colas. Puede verificar que la resincronización se ha llevado a cabo utilizando el mandato **rsvmqtrn** o el mandato de la propia base de datos para ver unidades de trabajo pendientes. Después de comprobar que la resincronización se ha llevado a cabo, finalice el gestor de colas y elimine la información de configuración de la base de datos.

Si no tiene en cuenta este procedimiento, el gestor de colas sigue recordando todas las unidades de trabajo pendientes correspondientes a dicha base de datos. Siempre que se reinicia el gestor de colas, se emite el mensaje de aviso AMQ7623. Si no va a configurar nunca más esa base de datos con el gestor de colas, indique al gestor de colas que olvide la participación de la base de datos en las transacciones pendientes utilizando la opción -r del mandato **rsvmqtrn**. El gestor de colas solamente olvida estas transacciones cuando las transacciones pendientes se hayan completado para todos los participantes.

En determinadas circunstancias puede necesitar eliminar temporalmente alguna información de configuración del gestor de recursos. En los sistemas UNIX and Linux, esto se consigue mejor comentando la stanza, de forma que se pueda reintegrar fácilmente más adelante. Puede que decida hacer esto si se producen errores cada vez que el gestor de colas se pone en contacto con una base de datos o un gestor de bases de datos determinado. Al eliminar temporalmente la información de configuración del gestor de recursos relativa permite al gestor de colas iniciar unidades de trabajo globales que impliquen a todos los demás participantes. El siguiente es un ejemplo de una sección `XAResourceManager` comentada:

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

Figura 17. Stanza XAResourceManager comentada en sistemas UNIX and Linux

En sistemas Windows , utilice WebSphere MQ Explorer para suprimir la información sobre la instancia del gestor de bases de datos. Asegúrese de escribir el nombre correcto en el campo *Name* cuando lo restablezca. Si no escribe bien el nombre, puede que tenga problemas con las unidades de trabajo pendientes, como se describe en "Modificación de la información de configuración" en la página 67.

Registro dinámico de XA

La especificación XA proporciona un método para reducir el número de llamadas `xa_*` que un gestor de transacciones realiza a un gestor de recursos. Esta optimización se conoce como *registro dinámico*.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

El registro dinámico está soportado por DB2. Otras bases de datos pueden darle soporte; consulte la documentación de su base de datos para obtener detalles.

¿Por qué es útil la optimización del registro dinámico? En su aplicación, algunas unidades de trabajo globales pueden contener actualizaciones para tablas de base de datos; otras puede que no las contengan. Cuando no se ha realizado ninguna actualización persistente en las tablas de una base de datos, no es necesario incluir dicha base de datos en el protocolo de confirmación que se lleva a cabo durante la ejecución de MQCMIT.

Tanto si la base de datos da soporte o no al registro dinámico, la aplicación llama a `xa_open` durante la primera llamada MQBEGIN en una conexión WebSphere MQ . También llama a `xa_close` en la posterior llamada MQDISC. El patrón de las llamadas XA posteriores depende de si la base de datos soporta o no el registro dinámico:

Si la base de datos no da soporte al registro dinámico...
Cada unidad de trabajo global implica varias llamadas de función XA realizadas por el código de WebSphere MQ en la biblioteca de cliente de base de datos, independientemente de si ha realizado una actualización persistente en las tablas de dicha base de datos dentro de la unidad de trabajo. Incluyen los siguientes:

- `xa_start` y `xa_end` desde el proceso de aplicaciones. Se utilizan para declarar el comienzo y el final de una unidad de trabajo global.
- `xa_prepare`, `xa_commit` y `xa_rollback` desde el proceso agente del gestor de colas, amqzlaa0. Se utilizan para entregar el resultado de la unidad de trabajo global: la decisión de confirmación o de restitución.

Además, el proceso del agente del gestor de colas también llama a `xa_open` durante el primer MQBEGIN.

Si la base de datos da soporte al registro dinámico...
El código de WebSphere MQ sólo realiza las llamadas de función XA que sean necesarias. Para una unidad de trabajo global que **no** incluya actualizaciones persistentes en recursos de base de datos, **no** habrá llamadas XA a la base de datos. Para una unidad de trabajo global que **sí** incluya dichas actualizaciones persistentes, las llamadas serán:

- `xa_end` desde el proceso de aplicaciones para declarar el final de la unidad de trabajo global.

- `xa_prepare`, `xa_commit` y `xa_rollback` desde el proceso agente del gestor de colas, amqzlaa0. Se utilizan para entregar el resultado de la unidad de trabajo global: la decisión de confirmación o de restitución.

Para que el registro dinámico funcione, es vital que la base de datos tenga una forma de indicar a WebSphere MQ cuándo ha realizado una actualización persistente que desea que se incluya en la unidad de trabajo global actual. WebSphere MQ proporciona la función `ax_reg` para esta finalidad.

El código de cliente de la base de datos que se ejecuta en el proceso de aplicaciones busca la función `ax_reg` y la llama, para *registrar dinámicamente* el hecho de haber hecho un trabajo persistente dentro de la unidad de trabajo global actual. En respuesta a esta llamada `ax_reg` , WebSphere MQ registra que la base de datos ha participado. Si esta es la primera llamada `ax_reg` en esta conexión de WebSphere MQ , el agente del gestor de colas llama a `xa_open`.

El código de cliente de base de datos realiza esta llamada `ax_reg` cuando se está ejecutando en el proceso, por ejemplo, durante una llamada SQL UPDATE o cualquier llamada de la que la API de cliente de la base de datos sea responsable.

Condiciones de error
En el registro dinámico de XA existe una posibilidad de anomalía confusa en el gestor de cola.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Un ejemplo frecuente es, si se olvida establecer las variables del entorno de la base de datos adecuadamente antes de iniciar el gestor de colas, las llamadas del entorno de colas a `xa_open` fallarán. No podrá utilizar unidades de trabajo globales.

Para evitar este problema, asegúrese de haber establecido las variables de entorno relevantes antes de iniciar el gestor de colas. Revise la documentación de su producto de base de datos y la información de los apartados “Configuración de Db2” en la página 52, “Configuración de Oracle” en la página 55 y “Configuración de Sybase” en la página 60.

Con todos los productos de base de datos, el gestor de colas llama a `xa_open` una vez durante el inicio del gestor de colas, como parte de la sesión de recuperación (como se explica en “Consideraciones cuando se pierde el contacto con el gestor de recursos XA” en la página 63). Esta llamada a `xa_open` no se ejecuta correctamente si se han establecido erróneamente las variables de entorno de la base de datos, pero no impide que el gestor de colas se inicie. Esto se debe a que la biblioteca de cliente de la base de datos utiliza el mismo código de error de `xa_open` para indicar que el servidor de bases de datos no se encuentra disponible. WebSphere MQ no trata esto como un error grave, ya que el gestor de colas debe poder empezar a continuar procesando datos fuera de las unidades de trabajo globales que implican a dicha base de datos.

Las llamadas posteriores a `xa_open` se realizan desde el gestor de colas durante el primer MQBEGIN en una conexión WebSphere MQ (si no se está utilizando el registro dinámico) o durante una llamada del código de cliente de base de datos a la función WebSphere MQproporcionada `ax_reg` (si se está utilizando el registro dinámico).

La **temporización** de las condiciones de error (o, ocasionalmente, de los informes FFST) depende de si está utilizando el registro dinámico:

- Si utiliza el registro dinámico, la llamada MQBEGIN podría realizarse satisfactoriamente pero la llamada de base de datos SQL UPDATE (o similar) fallará.
- Si no utiliza el registro dinámico, la llamada MQBEGIN fallará.

Asegúrese de que las variables de entorno se han establecido correctamente en los procesos del gestor de colas y de la aplicación.

Resumen de las llamadas XA
A continuación se muestra una lista de las llamadas realizadas a las funciones XA en una biblioteca de cliente de base de datos como resultado de las diversas llamadas MQI que controlan las unidades de

trabajo globales. No se trata de una descripción completa del protocolo descrito en la especificación de XA; se proporciona como una breve visión general.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Tenga en cuenta que las llamadas `xa_start` y `xa_end` siempre se invocan mediante el código WebSphere MQ en el proceso de aplicación, mientras que `xa_prepare`, `xa_commit`y `xa_rollback` siempre se invocan desde el proceso del agente del gestor de colas, amqzlaa0.

Las llamadas `xa_open` y `xa_close` que aparecen en esta tabla se realizan todas desde el proceso de aplicaciones. El proceso agente del gestor de colas llama a `xa_open` en las circunstancias que se describen en “Condiciones de error” en la página 69.

Tabla 7. Resumen de llamadas de función XA

Llamada MQI	**Llamadas XA realizadas con registro dinámico**	**Llamadas XA realizadas sin registro dinámico**
Primera MQBEGIN	`xa_open`	`xa_open` `xa_start`
MQBEGIN posterior	Ninguna llamada XA	`xa_start`

MQCMIT (**sin llamar a** `ax_reg` durante la unidad de trabajo global actual)	Ninguna llamada XA	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (**con** `ax_reg` que se llama durante la unidad de trabajo global actual)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		No aplicable. No se realizan llamadas a ax_reg en la modalidad no dinámica.

MQBACK (**sin que se llame a** `ax_reg` durante la unidad de trabajo global actual)	Ninguna llamada XA	`xa_end` `xa_rollback`
MQBACK (**con** `ax_reg` que se llama durante la unidad de trabajo global actual)	`xa_end` `xa_rollback`	No aplicable. No se realizan llamadas a ax_reg en la modalidad no dinámica.
MQDISC, donde se llamó a MQCMIT o MQBACK primero. Si no se les llamó, el proceso MQCMIT se realiza primero durante MQDISC.	`xa_close`	`xa_close`

Notas:

1. Para MQCMIT, se llama a `xa_commit` si `xa_prepare` se realiza correctamente. En caso contrario, se llama a `xa_rollback`.

Escenario 2: Otro software proporciona la coordinación

En el escenario 2, un gestor de transacciones externo coordina las unidades de trabajo globales, iniciándolas y confirmándolas bajo el control de la API del gestor de transacciones. Los verbos MQBEGIN, MQCMIT y MQBACK no están disponibles.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Esta sección describe este escenario e incluye:

- "Coordinación externa del punto de sincronización" en la página 71
- "Utilización de CICS" en la página 73
- "Utilización de Microsoft Transaction Server (COM +)" en la página 78

El cliente IBM WebSphere MQ para HP Integrity NonStop Server, puede utilizar HP NonStop Transaction Management Facility (TMF), para coordinar unidades globales de trabajo. Para obtener más información, consulte Utilización de HP NonStop TMF.

Coordinación externa del punto de sincronización

Una unidad de trabajo global también se puede coordinar mediante un gestor de transacciones externo compatible con XA de X/Open. Aquí el gestor de colas de WebSphere MQ participa en la unidad de trabajo, pero no la coordina.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

El flujo de control en una unidad de trabajo global coordinada por un gestor de transacciones externo es el siguiente:

1. Una aplicación indica al coordinador de punto de sincronización externo (por ejemplo, TXSeries) que desea iniciar una transacción.
2. El coordinador de puntos de sincronización indica a los gestores de recursos conocidos, como por ejemplo WebSphere MQ, sobre la transacción actual.
3. La aplicación emite llamadas a los gestores de recursos asociados a la transacción actual. Por ejemplo, la aplicación podría emitir llamadas de MQGET a WebSphere MQ.
4. La aplicación emite una solicitud de confirmación o restitución al coordinador externo del punto de sincronización.
5. El coordinador del punto de sincronización termina la transacción emitiendo las llamadas adecuadas a cada gestor de recursos, normalmente utilizando protocolos de confirmación en dos fases.

Los niveles soportados de coordinadores de puntos de sincronización externos que pueden proporcionar un proceso de confirmación en dos fases para las transacciones en las que participa WebSphere MQ se definen en IBM WebSphere MQ requisitos detallados del sistema.

En el resto de esta sección se describe cómo habilitar unidades de trabajo externas.

La estructura de conmutación de IBM WebSphere MQ XA

Cada gestor de recursos que participa en una unidad de trabajo coordinada externamente debe tener una estructura de conmutación de XA. Esta estructura define las posibilidades del gestor de recursos y las funciones a las que debe llamar el coordinador del punto de sincronización.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

IBM WebSphere MQ proporciona dos versiones de esta estructura:

- *MQRMIXASwitch* para la gestión de recursos XA estática.
- *MQRMIXASwitchDynamic* para la gestión de recursos XA dinámica.

Consulte la documentación del gestor de transacciones para determinar si utilizar o no la interfaz de gestión de recursos dinámica o estática. Siempre que un gestor de transacciones le dé soporte, es recomendable que se utilice la gestión de recursos XA dinámica.

Algunos gestores de transacciones de 64 bits tratan el tipo *largo* de la especificación XA como de 64 bits, y otros como de 32 bits. WebSphere MQ da soporte a ambos modelos:

- Si el gestor de transacciones es de 32 bits, o el gestor de transacciones es de 64 bits pero trata el tipo *largo* como de 32 bits, utilice el archivo de carga conmutada que se lista en la Tabla 8 en la página 72.
- Si el gestor de transacciones es de 64 bits y trata el tipo *long* como de 64 bits, utilice el archivo de carga conmutada que se lista en la Tabla 9 en la página 72.

En la Tabla 10 en la página 73 se proporciona una lista de gestores de transacciones conocidos de 64 bits que tratan el tipo *largo* como de 64 bits. Consulte la documentación del gestor de transacciones si no está seguro del modelo que utiliza el gestor de transacciones.

Tabla 8. Nombres de archivo de carga conmutada XA

Plataforma	**Nombre de archivo de carga conmutada (servidor)**	**Nombre de archivo de carga conmutada (cliente transaccional extendido)**
Windows	*mqmxa.dll*	*mqcxa.dll*
AIX (sin hebras)	*libmqmxa.a*	*libmqcxa.a*
AIX (con hebras)	*libmqmxa_r.a*	*libmqcxa_r.a*
HP-UX (sin hebras)	*libmqmxa.so*	*libmqcxa.so*
HP-UX (con hebras)	*libmqmxa_r.so*	*libmqcxa_r.so*
Linux (sin hebras)	*libmqmxa.so*	*libmqcxa.so*
Linux (con hebras)	*libmqmxa_r.so*	*libmqcxa_r.so*
Solaris	*libmqmxa.so*	*libmqcxa.so*

Tabla 9. Nombres de archivo de carga conmutada XA de 64 bits alternativos

Plataforma	**Nombre de archivo de carga conmutada (servidor)**	**Nombre de archivo de carga conmutada (cliente transaccional extendido)**
AIX (sin hebras)	*libmqmxa64.a*	*libmqcxa64.a*
AIX (con hebras)	*libmqmxa64_r.a*	*libmqcxa64_r.a*
HP-UX (sin hebras)	*libmqmxa64.so*	*libmqcxa64.so*
HP-UX (con hebras)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Linux (sin hebras)	*libmqmxa64.so*	*libmqcxa64.so*
Linux (con hebras)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Solaris	*libmqmxa64.so*	*libmqcxa64.so*

Tabla 10. Gestores de transacciones de 64 bits que requieren el archivo de carga conmutada de 64 bits alternativo
Gestor de transacciones
Tuxedo

Algunos coordinadores de puntos de sincronización externos (no CICS) requieren que cada gestor de recursos que participa en una unidad de trabajo proporcione su nombre en el campo de nombre de la estructura de conmutación XA. El nombre del gestor de recursos de WebSphere MQ es MQSeries_XA_RMI.

El coordinador de punto de sincronización define cómo se enlaza la estructura de conmutación XA de WebSphere MQ . La información sobre cómo enlazar la estructura de conmutación XA de WebSphere MQ con CICS se proporciona en "Utilización de CICS" en la página 73. Para obtener información sobre cómo enlazar la estructura de conmutación XA de WebSphere MQ con otros coordinadores de puntos de sincronización compatibles con XA, consulte la documentación que se proporciona con estos productos.

Las consideraciones siguientes se aplican a la utilización de WebSphere MQ con todos los coordinadores de punto de sincronización compatibles con XA:

- La estructura xa_info pasada en cualquier llamada xa_open por el coordinador de punto de sincronización incluye el nombre de un gestor de colas de WebSphere MQ . El nombre adopta el mismo formato que el nombre de gestor de colas que se pasa a la llamada `MQCONN`. Si el nombre pasado en la llamada xa_open está en blanco, se utiliza el nombre del gestor de colas predeterminado.

  De forma alternativa, la estructura xa_info puede contener valores para los parámetros *TPM* y *AXLIB*. El parámetro *TPM* especifica el gestor de transacciones que se va a utilizar. Los valores válidos son CICS, TUXEDO y ENCINA. El parámetro *AXLIB* especifica el nombre de la biblioteca que contiene las funciones ax_reg y ax_unreg del gestor de transacciones. Para obtener más información sobre estos parámetros, consulte Configuración de un cliente transaccional extendido. Si la estructura xa_info contiene cualquiera de estos parámetros, el nombre del gestor de colas se especifica en el parámetro *QMNAME*, a no ser que se utilice el gestor de colas predeterminado.
- Un solo gestor de colas puede participar cada vez en una transacción coordinada por una instancia de un coordinador externo del punto de sincronización. El coordinador del punto de sincronización está conectado efectivamente al gestor de colas y, por lo tanto, debe cumplir la norma que indica que sólo se da soporte a una conexión cada vez.
- Todas las aplicaciones que incluyen llamadas a un coordinador del punto de sincronización externo pueden conectarse únicamente al gestor de colas que está participando en la transacción gestionada por el coordinador externo (puesto que ya están conectadas efectivamente a dicho gestor de colas). No obstante, las aplicaciones de este tipo deben emitir una llamada `MQCONN` para obtener un manejador de conexión, y deben emitir una llamada `MQDISC` antes de salir.
- Un gestor de colas cuyas actualizaciones de recursos son coordinadas por un coordinador del punto de sincronización externo, debe iniciarse antes de que se inicie el coordinador externo del punto de sincronización. Del mismo modo, el coordinador del punto de sincronización debe terminar antes de que termine el gestor de colas.
- Si el coordinador del punto de sincronización externo finaliza de forma anormal, debe detener y reiniciar el gestor de colas ***antes*** de reiniciar el coordinador del punto de sincronización para asegurarse de que se resuelven correctamente todas las operaciones de mensajería no confirmadas en el momento de la anomalía.

Utilización de CICS

CICS es uno de los elementos de TXSeries.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Las versiones de TXSeries que son compatibles con XA (y utilizan un proceso de confirmación de dos fases) se definen en: IBM WebSphere MQ requisitos detallados del sistema

WebSphere MQ también da soporte a otros gestores de transacciones. Vaya a IBM WebSphere MQ requisitos detallados del sistema para ver la lista actual de software soportado.

Requisitos del proceso de confirmación en dos fases
Requisitos del proceso de confirmación de dos fases cuando se utiliza el proceso de confirmación de dos fases de CICS con WebSphere MQ. Estos requisitos no se aplican a z/OS.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Tenga en cuenta los requisitos siguientes:

- WebSphere MQ y CICS deben residir en la misma máquina física.
- WebSphere MQ no da soporte a CICS en un cliente MQI de WebSphere MQ .
- Debe iniciar el gestor de colas, con su nombre especificado en la stanza de definición de recursos XAD, ***antes*** de intentar iniciar CICS. Si no lo hace, no podrá iniciar CICS si ha añadido una stanza de definición de recurso XAD para WebSphere MQ a la región CICS .
- Solo se puede acceder a un gestor de colas de WebSphere MQ a la vez desde una única región CICS .
- Una transacción CICS debe emitir una solicitud `MQCONN` para poder acceder a los recursos de WebSphere MQ . La llamada `MQCONN` debe especificar el nombre del gestor de colas WebSphere MQ especificado en la entrada XAOpen de la stanza de definición de recurso XAD para la región CICS . Si esta entrada está en blanco, la solicitud `MQCONN` debe especificar el gestor de colas predeterminado.
- Una transacción CICS que accede a recursos WebSphere MQ debe emitir una llamada `MQDISC` desde la transacción antes de volver a CICS. Si no lo hace, podría significar que el servidor de aplicaciones CICS sigue conectado, dejando las colas abiertas. Además, si no instala una salida de terminación de tarea (consulte “Salida de terminación de tarea de ejemplo” en la página 77), es posible que el servidor de aplicaciones CICS finalice de forma anómala posteriormente, quizás durante una transacción posterior.
- Debe asegurarse de que el ID de usuario (cics) de CICS sea miembro del grupo mqm, para que el código CICS tenga autorización para llamar a WebSphere MQ.

  Para las transacciones que se ejecutan en un entorno CICS , el gestor de colas adapta sus métodos de autorización y determinación de contexto de la forma siguiente:

  – El gestor de colas consulta el ID de usuario bajo el que CICS ejecuta la transacción. Se trata del ID de usuario comprobado por el Gestor de autorizaciones sobre objetos y que se utiliza para la información de contexto.
  – En el contexto del mensaje, el tipo de aplicación es MQAT_CICS.
  – El nombre de aplicación en el contexto se copia del nombre de transacción CICS .

Soporte de XA general
General XA no está soportado en IBM i. Se proporciona un módulo de carga de conmutación XA para permitirle enlazar CICS con WebSphere MQ en sistemas UNIX and Linux . Además, se proporcionan archivos de código fuente de ejemplo para que pueda desarrollar los conmutadores XA para otros mensajes de transacciones.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Los nombres de los módulos de carga conmutada que se proporcionan son:

Tabla 11. Código esencial para aplicaciones CICS : rutina de inicialización XA	
C (fuente)	**C (exec) - añada uno de los siguientes a XAD.Stanza**
amqzscix.c	amqzsc - TXSeries para AIX, Versión 5.1, amqzsc - TXSeries para HP-UX, Versión 5.1 amqzsc - TXSeries for Sun Solaris, Versión 5.1
amqzscin.c	mqmc4swi - TXSeries para Windows, Versión 5.1

Creación de bibliotecas para utilizarlas con TXSeries for Multiplatforms
Utilice esta información al crear bibliotecas para utilizarlas con TXSeries for Multiplatforms.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Los *archivos de carga de conmutador precompilados* son bibliotecas compartidas (denominadas *DLL* en el sistema Windows) que puede utilizar con programas CICS, que requieren una transacción de confirmación de 2 fases utilizando el protocolo XA. Los nombres de estas bibliotecas precompiladas se encuentran en la tabla Código esencial para aplicaciones CICS : rutina de inicialización XA. También se proporciona código fuente de ejemplo en los directorios siguientes:

Tabla 12. Directorios de instalación en sistemas operativos Windows, UNIX and Linux		
Plataforma	**Directorio**	**Archivo de origen**
UNIX and Linux	*MQ_INSTALLATION_PATH*/ samp/	amqzscix.c
Windows	*MQ_INSTALLATION_PATH*\Tools \c \ Ejemplos	amqzscin.c

donde*MQ_INSTALLATION_PATH* es el directorio en el que ha instalado IBM WebSphere MQ.

Para compilar el archivo de carga conmutada a partir del código fuente de ejemplo, siga las instrucciones correspondientes a su sistema operativo:

AIX
Emita el mandato siguiente:

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

Solaris
Emita el mandato siguiente:

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

HP-UX

Emita el mandato siguiente:

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

Plataformas Linux

Emita el mandato siguiente:

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

Windows

Siga estos pasos:

1. Utilice el mandato cl para construir amqzscin.obj compilando al menos las variables siguientes:

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

2. Cree un archivo de definición de módulo llamado `mqmc1415.def` que contenga las líneas siguientes:

   ```
   LIBRARY MQMC4SWI
   EXPORTS
   CICS_XA_Init
   ```

3. Utilice el mandato **lib** para construir un archivo de exportación y una biblioteca de importación utilizando al menos la opción siguiente:

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

   Si la ejecución del mandato lib es satisfactoria, también se creará un archivo `mqmc4swi.exp`.

4. Utilice el mandato link para crear `mqmc4swi.dll` utilizando al menos la opción siguiente:

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

IBM WebSphere MQ Soporte XA y Tuxedo

IBM WebSphere MQ en Windows, los sistemasUNIX and Linux pueden bloquear aplicaciones XA coordinadas por Tuxedo indefinidamente en xa_start.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Esto sólo puede ocurrir cuando dos o más procesos coordinados por Tuxedo en un único intento de transacción global para acceder a IBM WebSphere MQ utilizando el mismo ID de rama de transacción (XID). Si Tuxedo proporciona a cada proceso de la transacción global un XID distinto para utilizarlo con IBM WebSphere MQ, esto no se puede producir.

Para evitar el problema, configure cada aplicación en Tuxedo que acceda a IBM WebSphere MQ bajo un único ID de transacción global (gtrid), dentro de su propio grupo de servidores Tuxedo. Los procesos del mismo grupo de servidores utilizan el mismo XID al acceder a los gestores de recursos en nombre de un único gtrid y, por lo tanto, son vulnerables al bloqueo en xa_start en IBM WebSphere MQ. Los procesos en

distintos grupos de servidores utilizan XID separados al acceder a los gestores de recursos y, por lo tanto, no tienen que serializar su trabajo de transacción en IBM WebSphere MQ.

Habilitación del proceso de confirmación en dos fases de CICS
Para permitir que CICS utilice un proceso de confirmación de dos fases para coordinar transacciones que incluyan llamadas MQI, añada una entrada de stanza de definición de recurso XAD de CICS a la región CICS . Tenga en cuenta que este tema no es aplicable a z/OS.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

A continuación se muestra un ejemplo de adición de una entrada de stanza XAD para WebSphere MQ para Windows, donde `<Drive>` es la unidad donde está instalado WebSphere MQ (por ejemplo, D:).

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

Para clientes transaccionales extendidos, utilice el archivo de carga conmutada mqcc4swi.dll.

A continuación se muestra un ejemplo de adición de una entrada de stanza XAD para sistemas WebSphere MQ for UNIX and Linux , donde *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ :

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

Para clientes transaccionales extendidos, utilice el archivo de carga de conmutación amqczsc.

Para obtener información sobre cómo utilizar el mandato **cicsadd** , consulte la publicación *CICS Administration Reference*o la publicación *CICS Administration Guide* correspondiente a su plataforma.

Las llamadas a WebSphere MQ se pueden incluir en una transacción CICS , y los recursos de WebSphere MQ se confirmarán o retrotraerán tal como indica CICS. Este soporte no está disponible para aplicaciones cliente.

debe emitir un `MQCONN` desde la transacción CICS para acceder a los recursos de WebSphere MQ , seguido de un `MQDISC` correspondiente al salir.

Habilitación de salidas de usuario de CICS
Un punto de salida de usuario de CICS (normalmente denominado *salida de usuario*) es un lugar en un módulo CICS en el que CICS puede transferir el control a un programa que ha escrito (un programa *de salida de usuario*) y en el que CICS puede reanudar el control cuando el programa de salida ha finalizado su trabajo.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Antes de utilizar una salida de usuario CICS , consulte la publicación *CICS Guía de administración* de la plataforma.

Salida de terminación de tarea de ejemplo
WebSphere MQ proporciona código fuente de ejemplo para una salida de terminación de tarea CICS .

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

El código fuente de ejemplo se encuentra en uno de los directorios siguientes:

Tabla 13. Salidas de terminación de tarea de CICS		
Plataforma	**Directorio**	**Archivo de origen**
Sistemas UNIX and Linux	*MQ_INSTALLATION_PATH*/samp	amqzscgx.c
Windows	*MQ_INSTALLATION_PATH*\Tools \c \ Ejemplos	amqzscgn.c

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Las instrucciones de compilación para la salida de terminación de tarea de ejemplo están contenidas en los comentarios cerca de la parte superior de cada archivo de origen.

Esta salida la invoca CICS en una terminación de tarea normal y anómala (después de que se haya tomado cualquier punto de sincronización). No se permite ningún trabajo recuperable en el programa de salida.

Estas funciones sólo se utilizan en un contexto de WebSphere MQ y CICS en el que la versión de CICS da soporte a la interfaz XA. CICS hace referencia a estas bibliotecas como "programas" o "salidas de usuario".

CICS tiene un número de salidas de usuario y amqzscgx, si se utiliza, se define y habilita en CICS como la "salida de usuario de terminación de tarea (UE014015)", es decir, la salida número 15.

Cuando CICSllama a la salida de terminación de tarea, CICS ya ha informado a WebSphere MQ del estado de terminación de la tarea y WebSphere MQ ha realizado la acción adecuada (confirmar o retrotraer). Lo único que hace la salida es emitir una llamada MQDISC para realizar una limpieza.

Una finalidad de instalar y configurar el sistema CICS para utilizar una salida de terminación de tarea es proteger el sistema frente a algunas de las consecuencias de un código de aplicación defectuoso. Por ejemplo, si la transacción CICS finaliza de forma anómala sin llamar primero a MQDISC y no tiene ninguna salida de terminación de tarea instalada, es posible que vea (en unos 10 segundos) una anomalía irrecuperable posterior de la región CICS . Esto se debe a que la hebra de salud de WebSphere MQ, que se ejecuta en el proceso cicsas, no se habrá publicado y no se le habrá dado tiempo para limpiar y devolver. Los síntomas pueden ser que el proceso cicsas finaliza inmediatamente, habiendo escrito FFST informes en /var/mqm/errors o la ubicación equivalente en Windows.

Utilización de Microsoft Transaction Server (COM +)

COM + (Microsoft Transaction Server) está diseñado para ayudar a los usuarios a ejecutar aplicaciones de lógica empresarial en un servidor de nivel medio típico.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Consulte Características que solo se pueden utilizar con la instalación primaria en Windows para obtener información importante.

COM + divide el trabajo en *Actividades*, que suelen ser fragmentos independientes de lógica empresarial, como por ejemplo *transferencia de fondos de la cuenta A a la cuenta B*. COM + se basa en gran medida en la orientación de los objetos y, en particular, en COM; una actividad de COM + está representada por un objeto COM (empresa).

COM+ es una parte integrada del sistema operativo. Para utilizar COM + en Windows 2000 y Windows XP, necesita Hotfix Q313582 (también conocido como COM + Rollup Package 19.1).

COM+ proporciona tres servicios al administrador de objetos de negocio, eliminando gran parte de los problemas del programador de objetos de negocio:

- Gestión de transacciones
- Seguridad

- Agrupación de recursos

Normalmente se utiliza COM + con código frontal que es un cliente COM para los objetos contenidos en COM + y servicios de fondo como, por ejemplo, una base de datos, con WebSphere MQ puente entre el objeto de negocio COM + y el programa de fondo.

El código frontal puede ser un programa autónomo o una ASP (Active Server Page) alojada por Microsoft Internet Information Server (IIS). El código frontal puede estar en el mismo sistema que COM + y sus objetos de negocio, con conexión a través de COM. Alternativamente, el código frontal puede estar en un ordenador diferente, con conexión a través de DCOM. Puede utilizar clientes diferentes para acceder al mismo objeto de negocio COM+ en situaciones diferentes.

El código de fondo puede estar en el mismo sistema que COM + y sus objetos de negocio, o en un sistema diferente con conexión a través de cualquiera de los protocolos soportados de WebSphere MQ .

Caducidad de las unidades de trabajo globales

El gestor de colas se puede configurar para que haga que las unidades de trabajo globales caduquen tras un intervalo de inactividad preconfigurado.

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

Para permitir este comportamiento, defina las siguientes variables de entorno:

- *AMQ_TRANSACTION_EXPIRY_RESCAN*= < intervalo rescan en milisegundos >
- *AMQ_XA_TRANSACTION_EXPIRE*= < intervalo de tiempo de espera en milisegundos >

Atención: Las variables de entorno sólo afectan a las transacciones que están en estado *Desocupado* en la tabla 6-4 de la especificación XA. Es decir, las transacciones que no están asociadas a ninguna hebra de aplicación, pero para las que el software gestor de transacciones externo todavía no ha llamado a la función **`xa_prepare`**.

Los gestores de transacciones externos conservan solamente un registro de las transacciones que se han preparado, confirmado y retrotraído. Si el gestor de transacciones externo se cuelga por algún motivo, al volverse a activar finalizará transacciones preparadas, confirmadas y retrotraídas pero aquellas transacciones activas que todavía no se han preparado se convierten en huérfanas. Para evitarlo, defina *AMQ_XA_TRANSACTION_EXPIRY* de forma que permita que un intervalo esperado entre una aplicación que realiza llamadas API transaccionales de MQI y la finalización de la transacción, haya terminado el trabajo transaccional en otros gestores de recursos.

Para garantizar una limpieza puntual después de que *AMQ_XA_TRANSACTION_EXPIRY* haya caducado, defina el valor *AMQ_TRANSACTION_EXPIRY_RESCAN* en un valor inferior al intervalo *AMQ_XA_TRANSACTION_EXPIRY*; lo ideal sería que se produjera una nueva exploración más de una vez durante el intervalo *AMQ_XA_TRANSACTION_EXPIRY*.

Unidad de disposición de recuperación

WebSphere MQ for z/OS proporciona disposiciones de unidad de recuperación. Esta función permite al usuario configurar si se puede dirigir la segunda fase de las transacciones de confirmación en dos fases, por ejemplo, durante la recuperación, cuando se conecta a otro gestor de colas dentro del mismo grupo de compartición de colas (QSG).

Nota: Este tema también está disponible en IBM MQ Version 8.0 y versiones posteriores. Sin embargo, no puede cambiar a una versión posterior utilizando el recuadro de lista "Cambiar versión" . Para ir al tema en una versión posterior, edite el número de versión en el recuadro de URL del navegador.

WebSphere MQ for z/OS V7.0.1 y posterior da soporte a la disposición de unidad de recuperación.

Unidad de disposición de recuperación

La disposición de la unidad de recuperación está relacionada con la conexión de una aplicación y posteriormente, con cualquier transacción que se inicie. Existen dos disposiciones de posibles de la unidad de recuperación.

- Una disposición de la unidad de recuperación GROUP identifica que una aplicación transaccional está conectada de forma lógica a un grupo de compartición de colas y no tiene afinidad con ningún gestor de colas específico. Las transacciones de confirmación en dos fases que se inician y que han completado la primera fase del proceso de confirmación, es decir, que están pendientes, pueden investigarse y resolverse al conectarse a los gestores de colas dentro de un grupo de compartición de colas (QSG). Esto significa que en un escenario de recuperación el coordinador de transacciones no tiene que volver a conectarse con el mismo gestor de colas, que puede no estar disponible.
- Una disposición de la unidad de recuperación QMGR identifica que una aplicación tiene una afinidad directa al gestor de colas al que está conectada y cualquier transacción que inicie tiene también esta disposición.

  En un escenario de recuperación el coordinador de transacciones debe volver a conectarse al mismo gestor de colas para preguntar y resolver las transacciones que están pendientes, independientemente de si el gestor de colas pertenece a un grupo de compartición de colas.

Decidir qué lenguaje de programación utilizar

Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

IBM WebSphere MQ proporciona soporte para los siguientes lenguajes de procedimiento de programación:

- C
- Visual Basic (solo sistemas Windows)
- COBOL

Estos idiomas utilizan la interfaz de cola de mensajes (MQI) para acceder a los servicios de colas de mensajes. Para obtener más información sobre el soporte para estos idiomas, consulte “Utilización de lenguajes de procedimiento con WebSphere MQ” en la página 80.

IBM WebSphere MQ proporciona soporte para:

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

Estos lenguajes utilizan el modelo de objetos IBM WebSphere MQ , que proporciona clases que proporcionan la misma funcionalidad que las llamadas y estructuras de WebSphere MQ , pero que son una forma más natural de programar en un entorno orientado a objetos. Algunos de los lenguajes que utilizan el modelo de objetos de IBM WebSphere MQ proporcionan funciones adicionales que no están disponibles en la interfaz de cola de mensajes (MQI). Para obtener más información sobre el soporte para estos idiomas, consulte “Programación orientada a objetos con WebSphere MQ” en la página 81.

Utilización de lenguajes de procedimiento con WebSphere MQ

Para obtener información detallada sobre cómo escribir las aplicaciones en el idioma elegido, consulte los enlaces siguientes:

- “Codificación en C” en la página 84
- “Codificación en Visual Basic” en la página 89
- “Desarrollo en COBOL” en la página 87

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.

WebSphere MQ proporciona archivos de definición de datos para ayudarle a escribir las aplicaciones. Para ver una descripción completa, consulte "IBM WebSphere MQ archivos de definición de datos" en la página 82.

Si puede elegir en qué idioma codificar los programas, tenga en cuenta la longitud máxima de los mensajes que procesarán los programas. Si los programas sólo procesarán mensajes de una longitud máxima conocida, puede codificarlos en cualquiera de los lenguajes de programación soportados. Pero si no conoce la longitud máxima de los mensajes que los programas tendrán que procesar, el lenguaje que elija dependerá de si está escribiendo un CICS, IMSo una aplicación 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
Codifique los programas en cualquier idioma soportado por CICS. La interfaz EXEC CICS proporciona las llamadas para gestionar la memoria, si es necesario.

Programación orientada a objetos con WebSphere MQ

Algunos de los lenguajes e infraestructuras de programación que utilizan el modelo de objetos de IBM WebSphere MQ proporcionan funciones adicionales que no están disponibles en la interfaz de cola de mensajes (MQI). Para obtener detalles de las clases, métodos y propiedades proporcionados por el modelo de objetos de IBM WebSphere MQ , consulte "El modelo de objeto de IBM WebSphere MQ" en la página 89.

.NET
Consulte Utilización de .NET para obtener información sobre cómo codificar programas .NET utilizando las clases .NET de WebSphere MQ . Message Service Clients for C/C++ and .NET proporcionan una interfaz de programación de aplicaciones (API) denominada XMS que tiene el mismo conjunto de interfaces que Java Message Service (JMS) API.

C++
IBM WebSphere MQ proporciona clases C++ equivalentes a objetos WebSphere MQ y algunas clases adicionales 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 WebSphere MQ Object Model en C++. Message Service Clients for C/C++ and .NET proporcionan una interfaz de programación de aplicaciones (API) denominada XMS que tiene el mismo conjunto de interfaces que Java Message Service (JMS) API.

Java
Consulte Utilización de Java para obtener información sobre cómo codificar programas utilizando el modelo de objetos WebSphere MQ en Java. Para obtener información sobre las diferencias entre las clases IBM WebSphere MQ para Java y las clases IBM WebSphere MQ para ayudarle a decidir qué utilizar, consulte "¿Debo utilizar clases IBM WebSphere MQ para Java o clases IBM WebSphere MQ para JMS?" en la página 91.

JMS
Websphere MQ también proporciona clases que implementan la especificación JMS (Java Message Service). Para obtener detalles de las clases de Websphere MQ para JMS, consulte Utilización de Java. Para obtener información sobre las diferencias entre las clases de IBM WebSphere MQ para Java y las clases de IBM WebSphere MQ para ayudarle a decidir qué utilizar, consulte "¿Debo utilizar clases IBM WebSphere MQ para Java o clases IBM WebSphere MQ para JMS?" en la página 91.

Message Service Clients for C/C++ and .NET proporcionan una interfaz de programación de aplicaciones (API) denominada XMS que tiene el mismo conjunto de interfaces que Java Message Service (JMS) API.

ActiveX
WebSphere MQ ActiveX se conoce comúnmente como MQAX. MQAX se incluye como parte de WebSphere MQ para Windows. El soporte para ActiveX se ha estabilizado en el nivel WebSphere MQ Versión 6.0 . Para aprovechar las características introducidas en WebSphere MQ posterior a la

versión 6.0, considere la posibilidad de utilizar .NET en su lugar. Consulte Utilización de la interfaz del Modelo de objetos componentes (WebSphere MQ Automation Classes for ActiveX) para obtener información sobre la codificación de programas utilizando el modelo de objetos WebSphere MQ en ActiveX.

Conceptos relacionados

Visión general técnica

“Desarrollo de aplicaciones” en la página 7

IBM WebSphere MQ proporciona varias formas en las que puede desarrollar aplicaciones para enviar y recibir mensajes que necesita para dar soporte a los procesos de negocio. También puede desarrollar aplicaciones para gestionar los gestores de colas y recursos relacionados.

“Conceptos de desarrollo de aplicaciones” en la página 8

Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

Referencia relacionada

Referencia para el desarrollo de aplicaciones

IBM WebSphere MQ archivos de definición de datos

IBM WebSphere 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 deWebSphere 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 WebSphere MQ” en la página 382.

Para los archivos de definición de datos soportados en C++, consulte Utilización de C++.

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 calificador de alto nivel de la biblioteca de instalación en z/OS.

En este tema se presentan los archivos de definición de datos de WebSphere MQ , bajo estas cabeceras:

- "Archivos include (de inclusión) del lenguaje C" en la página 83
- "Archivos de módulo Visual Basic" en la página 83
- "Archivos de copia COBOL" en la página 83

Archivos include (de inclusión) del lenguaje C

Los archivos de inclusión C de WebSphere MQ se listan en Archivos de cabecera C. Se instalan en los directorios o las bibliotecas siguientes:

Plataforma	Directorio o biblioteca de instalación
Plataformas UNIX	*MQ_INSTALLATION_PATH*/inc/
Sistemas Windows	*MQ_INSTALLATION_PATH*\Herramientas \c\include

donde *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Nota: Para las plataformas UNIX , los archivos de inclusión se enlazan simbólicamente en `/usr/include`.

Para obtener más información sobre la estructura de directorios, consulte Planificación del soporte del sistema de archivos .

Archivos de módulo Visual Basic

WebSphere MQ para 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, WebSphere 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).

Los archivos de copia COBOL de WebSphere MQ se listan en Archivos COPY de COBOL. Se instalan en los directorios siguientes:

Plataforma	Directorio o biblioteca de instalación
Otras plataformas UNIX	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (para Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (para IBM VisualAge COBOL)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere 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.

Codificación en C

Tenga en cuenta la información de las secciones siguientes al codificar programas WebSphere MQ en C.

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, la 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. Por ejemplo, el símbolo ¬ representa un 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
```

Desarrollo en COBOL

Tenga en cuenta la información de la sección siguiente al codificar programas WebSphere 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 WebSphere MQ que se pasan para iniciar la llamada MQ estén alineadas en los límites de palabra. Un límite de palabra es de 4 bytes en procesos de 32 bits, 8 bytes en procesos de 64 bits y 16 bytes en procesos de 128 bits (IBM i).

En la medida de lo posible, coloque todas las estructuras IBM WebSphere MQ juntas para que estén alineadas por límite.

Desarrollo en pTAL

Tenga en cuenta la información de la siguiente sección cuando desarrolle programas IBM WebSphere MQ en pTAL.

HP Integrity NonStop Server

Definición e inicialización de estructuras de IBM WebSphere MQ

Las definiciones de la estructura pTAL para IBM WebSphere MQ se proporcionan con nombres que finalizan con ^DEF. Por ejemplo, se codificarían las siguientes declaraciones pTAL para crear una estructura de descriptor de mensaje IBM WebSphere MQ (MQ Message Descriptor, MQMD) y una estructura de opciones de colocación de mensaje IBM WebSphere MQ (Put Message Options, MQPMO).

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

IBM WebSphere MQ proporciona pTAL DEFINE con nombres que terminan con ^DEFAULT para inicializar estructuras IBM WebSphere MQ con valores predeterminados. Se codifican las siguientes sentencias pTAL para asignar valores predeterminados a las estructuras MQMD y MQPMO declaradas:

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

Se pueden declarar e inicializar otras estructuras de IBM WebSphere MQ utilizando un código similar.

pTAL y el CRE

Los programas pTAL no pueden inicializar el entorno de ejecución común (Common Runtime Environment, CRE) y, por tanto, hay que usarlos con una rutina principal en C o en COBOL.

Los ejemplos de pTAL que se proporcionan con IBM WebSphere MQ utilizan una rutina de secuencia principal en C que se llama AMQSPTM0.C

Parámetros con tipo de datos MQCHAR

Los procedimientos MQGET, MQPUT y MQPUT1 tienen cada uno un parámetro **Buffer** con tipo de datos MQCHAR .EXT. 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 pTAL como vectores de cadenas. 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 del procedimiento se declara como un MQCHAR .EXT, pero la dirección de cualquier dato se puede especificar como parámetro en la invocación del procedimiento.

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.

Codificación en Visual Basic

Tenga en cuenta la información de la sección siguiente al codificar programas WebSphere MQ en Visual Basic.

Nota: Fuera del entorno .NET, el soporte para Visual Basic (VB) en WebSphere MQ se ha estabilizado en el nivel V6.0 . La mayoría de las funciones nuevas añadidas a WebSphere MQ 7.0 o posterior no están disponibles para las aplicaciones VB. Si está programando en VB.NET, utilice las clases .NET de WebSphere MQ . Para obtener más información, consulte Utilización de .NET.

Visual Basic sólo está soportado en Windows.

Para evitar la conversión no deseada de datos binarios que pasan entre Visual Basic y WebSphere MQ, utilice una definición MQBYTE en lugar de MQSTRING. CMQB.BAS define varios tipos MQBYTE nuevos que son equivalentes a una definición de byte C y los utiliza en estructuras WebSphere 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 WebSphere MQ son por desplazamiento en lugar de por puntero. Declare una estructura compuesta formada por las dos estructuras de componentes y especifique la estructura compuesta en la llamada. El soporte de WebSphere MQ para Visual Basic proporciona una llamada MQCONNXAny para hacerlo posible y permitir que las aplicaciones cliente especifiquen 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 un tipo de datos Cualquiera en lugar de un 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.

El modelo de objeto de IBM WebSphere MQ

El modelo de objeto de IBM WebSphere MQ consta de clases, métodos y propiedades. Utilice esta información para aprender sobre cada uno de estos conceptos.

El modelo de objeto IBM WebSphere MQ consta de lo siguiente:

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

Al crear una aplicación WebSphere MQ utilizando el modelo de objetos WebSphere MQ , se crean instancias de estas clases en el programa. 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).

Estos temas describen detalladamente cada uno de los modelos de objetos de WebSphere MQ :

- "Clases" en la página 89
- "Referencias de objetos" en la página 90
- "Códigos de retorno" en la página 91

Clases

El modelo de objeto WebSphere 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.

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.

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 WebSphere MQ que utiliza la MQI, WebSphere MQ devuelve manejadores de conexión y manejadores de objeto al programa.

Estos descriptores de contexto deben pasarse como parámetros en las llamadas posteriores a WebSphere MQ . Con el modelo de objeto WebSphere 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.

¿Debo utilizar clases IBM WebSphere MQ para Java o clases IBM WebSphere MQ para JMS?

Una aplicación Java puede utilizar clases IBM WebSphere MQ para Java o clases IBM WebSphere MQ para JMS para acceder a recursos IBM WebSphere MQ . Cada propuesta tiene sus ventajas.

IBM WebSphere MQ classes for Java encapsula la interfaz de cola de mensajes (MQI), la API nativa de IBM WebSphere MQ , y utiliza el mismo modelo de objeto que otras interfaces orientadas a objetos, mientras que IBM WebSphere MQ classes for Java Message Service implementa las interfaces JMS (Java Message Service) de Sun.

Si está familiarizado con IBM WebSphere MQ en entornos distintos de Java, utilizando lenguajes de procedimiento u orientados a objetos, puede transferir los conocimientos existentes al entorno Java utilizando clases IBM WebSphere MQ para Java. También puede aprovechar el rango completo de características de IBM WebSphere MQ, no todas las cuales están disponibles en las clases IBM WebSphere MQ para JMS.

Si no está familiarizado con IBM WebSphere MQ, o ya tiene experiencia en JMS, es posible que le resulte más fácil utilizar la API JMS familiar para acceder a los recursos IBM WebSphere MQ , utilizando clases IBM WebSphere MQ para JMS. JMS también forma parte integral de la plataforma Java Platform, Enterprise Edition (Java EE). Las aplicaciones Java EE pueden utilizar beans controlados por mensajes (MDB) para procesar mensajes de forma asíncrona, y los MDB solo pueden procesar mensajes JMS. JMS también es el mecanismo estándar para que Java EE interactúe con sistemas de mensajería asíncrona como IBM WebSphere MQ. Cada servidor de aplicaciones que sea compatible con Java EE debe incluir un proveedor JMS, por lo tanto, puede utilizar JMS para comunicarse entre distintos servidores de aplicaciones o puede portar una aplicación de un proveedor JMS a otro sin realizar ningún cambio en la aplicación.

Diseño de aplicaciones IBM WebSphere MQ

Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

Durante el diseño de una aplicación IBM WebSphere 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 WebSphere MQ. Para obtener más información, consulte Introducción a WebSphere MQ Administration Interface (MQAI) y Automatización de tareas de administración .

Lenguaje de programación

IBM WebSphere MQ soporta una serie de lenguajes de programación de procedimiento y orientada a objetos para escribir aplicaciones. Para obtener más información, consulte “Decidir qué lenguaje de programación utilizar” en la página 80.

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.

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 soportadas, consulte 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. Consulte Clústeres de gestores de colas para obtener más información.

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 “Diseño de los mensajes” en la página 94.

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 WebSphere MQ a un destino estándar gestionado mediante la publicación/suscripción de IBM WebSphere MQ y permite que IBM WebSphere 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 Introducción a la mensajería de publicación/suscripción de IBM WebSphere MQ .

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.

Control de los programas IBM WebSphere 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 WebSphere MQ *desencadenamiento* , consulte “Inicio de aplicaciones de IBM WebSphere MQ utilizando desencadenantes” en la página 337). 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 WebSphere MQ *sucesos de instrumentación* tal como se describe en Sucesos de instrumentación).

Ejecución de la aplicación en un cliente IBM WebSphere MQ

La MQI completa está soportada en el entorno de cliente y esto permite que casi cualquier aplicación IBM WebSphere MQ se vuelva a enlazar para que se ejecute en un cliente MQI de IBM WebSphere MQ . Enlace la aplicación en el cliente MQI de IBM WebSphere MQ a la biblioteca MQIC, en lugar de a la biblioteca MQI.

Nota: Una aplicación que se ejecuta en un cliente de IBM WebSphere 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 cliente MQI de IBM WebSphere MQ” en la página 368 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 WebSphere MQ, consulte la sección “Diseño y rendimiento de aplicaciones” en la página 95 .

Técnicas avanzadas de IBM WebSphere MQ

Para aplicaciones más avanzadas, puede ser conveniente utilizar algunas técnicas avanzadas de IBM WebSphere MQ tales como la correlación de respuestas y la generación y envío de información de contexto de IBM WebSphere MQ. Para obtener más información, consulte “Técnicas avanzadas de IBM WebSphere MQ” en la página 96.

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 WebSphere 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 330 para obtener más detalles). Puede utilizar la característica de *persistencia* de los mensajes de IBM WebSphere MQ para asegurar la entrega de mensajes importantes.

Prueba de aplicaciones IBM WebSphere MQ

El entorno de desarrollo de aplicaciones para programas IBM WebSphere 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 WebSphere MQ.

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

IBM WebSphere MQ

“Conceptos de desarrollo de aplicaciones” en la página 8
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

“Escritura de una aplicación de gestión de colas” en la página 199
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 cliente” en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

“Utilización de .NET” en la página 571

WebSphere MQ classes for .NET permite a un programa escrito en la infraestructura de programación .NET conectarse a WebSphere MQ como un cliente MQI de WebSphere MQ o conectarse directamente a un servidor WebSphere MQ .

“Utilización de C++” en la página 641
WebSphere MQ proporciona clases C++ equivalentes a los objetos WebSphere MQ y algunas clases adicionales equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

“Utilización de clases de WebSphere MQ para JMS” en la página 729
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) es el proveedor JMS que se proporciona con WebSphere MQ. Además de implementar las interfaces definidas en el paquete javax.jms , WebSphere MQ classes for JMS proporciona dos conjuntos de extensiones para la API JMS.

“Utilización de clases de WebSphere MQ para Java” en la página 666
WebSphere MQ classes for Java le permite utilizar WebSphere MQ en un entorno Java. Una aplicación Java puede utilizar clases WebSphere MQ para Java o clases WebSphere MQ para JMS para acceder a recursos WebSphere MQ .

“Utilización de la interfaz de modelo de objeto de componente (WebSphere MQ Automation Classes for ActiveX)” en la página 1052
Las clases de automatización de WebSphere MQ para ActiveX (MQAX) son componentes ActiveX que proporcionan clases que puede utilizar en la aplicación para acceder a WebSphere MQ.

Diseño de los mensajes

Tenga en cuenta estaas consideraciones a la hora de diseñar mensajes.

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 persistentes, recuperándolos cuando sea necesario desde los archivos de registro de WebSphere 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 WebSphere MQ para Windows o WebSphere MQ en sistemas UNIX 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 se ejecutará hasta los límites del archivo de registro.

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
Si es posible que sea necesario colocar un mensaje en varias colas, puede utilizar una lista de distribucióno publicar en un tema.

Diseño y rendimiento de aplicaciones

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 de los programas que realizan llamadas de WebSphere MQ .

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 las va a reutilizar en lugar de abrir y cerrar repetidamente, conectar y desconectar.
- 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 los mensajes dentro de una unidad de trabajo, para que se puedan confirmar o restituir simultáneamente.
- 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.

Búsqueda de un mensaje determinado

La llamada MQGET suele recuperar el primer mensaje de una cola. Si utiliza los identificadores de mensaje y 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 que encuentre 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 la aplicación para que el almacenamiento intermedio se redimensione en consecuencia y se vuelva a emitir la llamada MQGET.

Nota: Si no establece el atributo *MaxMsgLength* de forma explícita, el valor predeterminado es 4 MB, lo que puede ser muy ineficiente si se utiliza para influir en el tamaño del almacenamiento intermedio de la 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

Para WebSphere MQ para Windows, una aplicación puede requerir un gran número 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.

Técnicas avanzadas de IBM WebSphere MQ

Para una aplicación IBM WebSphere MQ simple, debe decidir qué objetos de WebSphere MQ utilizar en la aplicación y qué tipos de mensajes 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 274).
- Estableciendo una salida de devolución de llamada (callback) que se accione cuando llegue un mensaje; consulte “Consumo asíncrono de mensajes IBM WebSphere MQ” en la página 34.
- 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 aplicaciones WebSphere MQ , cuando un programa recibe un mensaje que le solicita que realice algún trabajo, el programa normalmente envía 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 39.

Inicio automático de programas WebSphere MQ

Utilice WebSphere 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 WebSphere MQ utilizando desencadenantes” en la página 337. Los desencadenantes no son más que una forma de iniciar un programa automáticamente. Por ejemplo, puede iniciar un programa automáticamente en un temporizador utilizando recursos que no son deWebSphere MQ .

WebSphere MQ puede definir objetos de servicio para iniciar programas WebSphere MQ cuando se inicia el gestor de colas; consulte Objetos de servicio.

Generación de informes de WebSphere 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).

Estas normas se describen en “Mensajes de informe” en la página 11.

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 afinidades de mensajes .

Programas WebSphere MQ de ejemplo

Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

- “Programas de ejemplo para plataformas distribuidas” en la página 98

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 8
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

“Decidir qué lenguaje de programación utilizar” en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

“Diseño de aplicaciones IBM WebSphere MQ” en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

“Escritura de una aplicación de gestión de colas” en la página 199
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 cliente” en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

“Utilización de servicios web en WebSphere MQ” en la página 965
Puede desarrollar aplicaciones IBM WebSphere MQ para servicios web utilizando el transporte IBM WebSphere MQ para SOAP o el puente IBM WebSphere MQ para HTTP.

“Escritura de aplicaciones de publicación/suscripción” en la página 284
Empiece a escribir aplicaciones de publicación/suscripción de WebSphere MQ .

“Creación de una aplicación IBM WebSphere MQ” en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

“Manejo de errores de programa” en la página 559
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.

Programas de ejemplo para plataformas distribuidas

En este tema se describen los programas de ejemplo que se entregan con IBM WebSphere MQ, escritos en C y COBOL. Los ejemplos muestran los usos típicos de la interfaz de cola de mensajes (MQI).

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. Sin embargo, estos ejemplos son adecuados para su uso como base para sus propios programas de colas de mensajes.

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.

Programas de ejemplo C++: Consulte Utilización de C++ para obtener una descripción de los programas de ejemplo disponibles en C++.

Los nombres de los ejemplos empiezan por el prefijo `amq`. El cuarto carácter indica el lenguaje de programación y el compilador donde 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 configurar el gestor de colas para aceptar conexiones de cliente, consulte “Preparación y ejecución de los programas de ejemplo” en la página 112 para obtener detalles.

Utilice los enlaces siguientes para obtener más información sobre los programas de ejemplo:

- “Características demostradas en los programas de ejemplo” en la página 100
- “Los programas de ejemplo de publicación/suscripción” en la página 138
- “Programas de transferencia de ejemplo” en la página 144
- “El programa de ejemplo de lista de distribución” en la página 131
- “Los programas de ejemplo de examen” en la página 119
- “El programa de ejemplo del navegador” en la página 120
- “Programas de ejemplo de obtención” en la página 132
- “Programas de ejemplo de mensaje de referencia” en la página 145
- “Programas de ejemplo de solicitud” en la página 151
- “Programas de ejemplo de consulta” en la página 137
- “Programa de ejemplo de consulta de propiedades de un manejador de mensajes” en la página 138
- “Programas Set de ejemplo” en la página 155
- “Los programas de ejemplo de eco” en la página 132
- “El programa de ejemplo de conexión de datos” en la página 123
- “Programas de ejemplo de desencadenamiento” en la página 159
- “El programa de ejemplo Asynchronous Put (operación de transferencia asíncrona)” en la página 118
- “Ejemplos de coordinación de bases de datos” en la página 123
- “El ejemplo de transacción CICS” en la página 121
- “ejemplos de TUXEDO” en la página 160
- “Ejemplo de cola de mensajes no entregados” en la página 130
- “El programa de ejemplo de conexión” en la página 121
- “El programa de ejemplo de salida de API” en la página 116
- “Utilización de la salida de seguridad SSPI en sistemas Windows” en la página 173
- “Ejecución de los ejemplos utilizando colas remotas” en la página 174

- "El programa de ejemplo Cluster Queue Monitoring (AMQSCLM)" en la página 174
- "Programa de ejemplo para Connection Endpoint Lookup (CEPL)" en la página 184

Características demostradas en los programas de ejemplo

Colección de tablas que muestran las técnicas demostradas por los programas de ejemplo de WebSphere 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.

Ejemplos para sistemas UNIX and Linux

Este tema muestra las técnicas demostradas por los programas de ejemplo para WebSphere MQ en sistemas UNIX and Linux .

Consulte "Preparación y ejecución de programas de ejemplo en sistemas UNIX" en la página 114 para averiguar dónde se almacenan los programas de ejemplo para WebSphere MQ en sistemas UNIX y Linux .

Tabla 14 en la página 100 La tabla lista qué archivos fuente C y COBOL se proporcionan, y si se incluye un ejecutable de servidor o cliente.

Tabla 14. Programas de ejemplo de WebSphere MQ en UNIX and Linux que demuestran el uso de MQI (C y COBOL)

Técnica	**C (fuente) ("1" en la página 103)**	**COBOL (fuente) ("2" en la página 103)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C) ("3" en la página 103)**
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 ("4" en la página 103)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Respuesta a un mensaje de solicitud	amqsinqa	amqminqx amqiinqx	amqsinq	no hay ejemplo
Obtención de mensajes (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 ("4" en la página 103)	amqsprma	no hay ejemplo	amqsprm	amqsprmc

Tabla 14. Programas de ejemplo de WebSphere MQ en UNIX and Linux que demuestran el uso de MQI (C y COBOL) (continuación)

Técnica	**C (fuente) ("1" en la página 103)**	**COBOL (fuente) ("2" en la página 103)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C) ("3" en la página 103)**
Obtención de mensajes de referencia de una cola ("4" en la página 103)	amqsgrma	no hay ejemplo	amqsgrm	amqsgrmc
Salida de canal de mensajes de referencia ("4" en la página 103)	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
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	("5" en la página 103)	no hay ejemplo	no hay ejemplo	no hay ejemplo
WebSphere MQ (coordinación de gestores de bases de datos compatibles con XA) que acceden a una única base de datos utilizando SQL	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	no hay ejemplo	no hay ejemplo
WebSphere MQ (coordinación de gestores de bases de datos compatibles con XA) que acceden a dos bases de datos utilizando SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	no hay ejemplo	no hay ejemplo
Transacción CICS ("6" en la página 103)	amqscic0.ccs	no hay ejemplo	amqscic0	no hay ejemplo
Transacción Encina ("4" en la página 103)	amqsxae0	no hay ejemplo	amqsxae0	no hay ejemplo

Tabla 14. Programas de ejemplo de WebSphere MQ en UNIX and Linux que demuestran el uso de MQI (C y COBOL) (continuación)

Técnica	**C (fuente) ("1" en la página 103)**	**COBOL (fuente) ("2" en la página 103)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C) ("3" en la página 103)**
Transacción TUXEDO para colocar mensajes ("7" en la página 103)	amqstxpx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Transacción TUXEDO para obtener mensajes ("7" en la página 103)	amqstxgx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Servidor de TUXEDO ("7" en la página 103)	amqstxsx	no hay ejemplo	no hay ejemplo	no hay ejemplo
Manejador de cola de mensajes no entregados	Directorio `./tools/c/Samples/dlq` ("8" en la página 103)	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 información de conexión SSL/TLS en MQCONNX	amqssslc	no hay ejemplo	no aplicable	amqssslc

Tabla 14. Programas de ejemplo de WebSphere MQ en UNIX and Linux que demuestran el uso de MQI (C y COBOL) (continuación)

Técnica	C (fuente) ("1" en la página 103)	COBOL (fuente) ("2" en la página 103)	Servidor (ejecutable C)	Cliente (ejecutable C) ("3" en la página 103)

Notas:

1. La versión ejecutable de los ejemplos de cliente MQI de WebSphere MQ comparten el mismo origen que los ejemplos que se ejecutan en un entorno de servidor.
2. Compile los programas que empiecen por 'amqm' con el compilador Micro Focus COBOL, los que empiecen por 'amqi' con el compilador COBOL de IBM y los que empiecen por 'amq0' con cualquiera de ellos.
3. Las versiones ejecutables de los ejemplos de cliente MQI de WebSphere MQ no están disponibles en WebSphere MQ para HP-UX.
4. Soportado en WebSphere MQ para AIX, WebSphere MQ para HP-UXy WebSphere MQ para Solaris solamente.
5. En WebSphere MQ para AIX, WebSphere MQ para HP-UXy WebSphere MQ para Solaris este programa se denomina `amqsvfc0.c`
6. CICS solo está soportado por WebSphere MQ para AIX y WebSphere MQ para HP-UX .
7. TUXEDO no está soportado por WebSphere MQ para Linux en System p.
8. El origen del manejador de colas de mensajes no entregados consta de varios archivos y se proporciona en un directorio aparte.

Encontrará información detallada sobre el soporte para sistemas UNIX and Linux en la página de requisitos de sistemas WebSphere MQ en Requisitos del sistema para IBM WebSphere MQ.

Ejemplos para el cliente de IBM WebSphere MQ para HP Integrity NonStop Server

En este tema se muestran las técnicas demostradas por los programas de ejemplo para el cliente IBM WebSphere MQ en sistemas HP Integrity NonStop Server .

Tabla 15 en la página 103La tabla lista los programas de ejemplo de origen C, COBOL y pTAL que se proporcionan.

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL

Técnica	C				COBOL		pTAL	
	OSS (Fuente)	OSS (Ejecutable)	Guardian (Fuente)	Guardian (Ejecutable)	OSS (Fuente)	Guardian (Fuente)	OSS (Fuente)	Guardian (Fuente)
Utilización de la interfaz de publicación/ suscripción	amqspuba.c amqssbxa.c amqssuba.c amqspse0.c	amqspubc amqssbxc amqssubc	MQSPUBC MQSSBXC MQSSUBC	AMQSPUBC AMQSSBXC AMQSSUBC	amq0pub0.cbl amq0sub0.cbl	MQSPUBL MQSSUBL	amqtpub0.tal amqtsub0.tal	MQSPUBT MQSSUBT

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	**C**				**COBOL**		**pTAL**	
Colocación de mensajes mediante la llamada MQPUT	amqsput0.c	amqsputc	MQSPUTC	AMQSPUTC	amq0put0.cbl	MQSPUTL	amqtput0.tal	MQSPUTT
Colocación de un único mensaje utilizando la llamada MQPUT1	amqsecha.c	amqsechc	MQSECHC	AMQSECHC			amqtech0.tal	MQSECHT
Colocación de mensajes en una lista de distribución	amqsptl0.c	amqsptlc	MQSPTLC	AMQSPTLC	amq0ptl0.cbl	MQSPTLL		
Respuesta a un mensaje de solicitud	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Obtención de mensajes (sin espera)	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Obtención de mensajes (espera con límite de tiempo)	amqsget0.c	amqsgetc	MQSGETC	AMQSGETC	amq0get0.cbl	MQSGETL	amqtget0.tal	MQSGETT
Obtención de mensajes (espera ilimitada)	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	**C**				**COBOL**		**pTAL**	
Obtención de mensajes (con conversión de datos)	amqsecha.c	amqsechc	MQSECHC	AMQSECHC				
Colocación de mensajes de referencia en una cola	amqsprma.c	amqsprmc	MQSPRMC	AMQSPRMC				
Obtención de mensajes de referencia de una cola	amqsgrma.c	amqsgrmc	MQSGRMC	AMQSGRMC				
Salida de canal de mensaje de referencia	amqsqrma.c amqsxrma.c		MQSQRMC MQSXRMC					
Exploración de los primeros 20 caracteres de un mensaje	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Exploración de mensajes completos	amqsbcg0.c	amqsbcgc	MQSBCGC	AMQSBCGC				
Utilización de una cola de entrada compartida	amqsinqa.c	amqsinqc	MQSINQC	MQSINQC				

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	**C**				**COBOL**		**pTAL**	
Utilización de una cola de entrada exclusiva	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
Utilización de la llamada MQINQ	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Utilización de la llamada MQSET	amqsseta.c	amqssetc	MQSSETC	AMQSSETC				
Utilización de una cola de respuesta	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
Solicitud de excepciones de mensaje	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
Aceptación de un mensaje truncado	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Utilización de un nombre de cola resuelto	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Desencadenamiento de un proceso	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
Utilización de la conversión de datos	amqsvfc0.c							

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	**C**				**COBOL**		**pTAL**	
Manejador de cola de mensajes no entregados (1)	Directorio ./ samp/dlq							
Conexión con un gestor de colas utilizando MQCONNX	amqscnxc.c	amqscnxc	MQSCNXC					
Utilización de salidas de API	amqsaxe0.c amqsaem0.c							
Salida de equilibrado de cargas de trabajo en un clúster	amqswlm0.c		MQSWLMC					
Supervisor de colas de clúster	amqsclma.c							
Colocación de mensajes de forma asíncrona y obtención del estado con la llamada MQSTAT	amqsapt0.c	amqsaptc	MQSAPTC	MQSAPTC				

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	**C**				**COBOL**		**pTAL**	
Clientes que se pueden volver a conectar	amqsghac.c amqsmhac.c amqsphac.c	amqsghac amqsmhac amqsphac	MQSGHAC MQSMHAC MQSPHAC MQSFHAC	AMQSGHAC AMQSMHAC AMQSPHAC AMQSFHAC				
Utilización de consumidores de mensajes para consumir mensajes de varias colas asíncronamente	amqscbf0.c	amqscbfc						
Especificación de información de conexión SSL/TLS en MQCONNX	amqssslc.c	amqssslc	MQSSSLC	AMQSSSLC				
Rastreo de actividades	amqsact0.c	amqsactc	MQSACTC	AMQSACTC				
Propiedades del mensaje	amqsiqma.c amqsstma.c	amqsiqmc amqsstmc	MQSIQMC MQSSTMC	AMQSIQMC AMQSSTMC				
Servidor de mandatos	amqsstop.c		MQSSTOC					
Eventos de anotación cronológica	amqslog0.c	amqslogc	MQSLOGC	AMQSLOGC				

Tabla 15. Programas de ejemplo de IBM WebSphere MQ en HP Integrity NonStop Server que ilustran el uso de C, COBOL y pTAL (continuación)

Técnica	C				COBOL		pTAL	
Contabilidad.	amqsmon0.c	amqsmonc	MQSMONC	AMQSMONC				
Interfaz de administración	amqsaicq.c amqsaiem.c amqsailq.c							
Ejemplo de función main en lenguaje C para invocar pTAL			MQSPTMC					

Notas:

1. El origen del manejador de colas de mensajes no entregados consta de varios archivos y se proporciona en un directorio aparte.
2. Para obtener información sobre el desarrollo de aplicaciones para su cliente IBM WebSphere MQ en la plataforma HP Integrity NonStop Server, consulte:
   - "Creación de la aplicación en HP Integrity NonStop Server" en la página 444
     - "Preparación de programas C en HP Integrity NonStop Server" en la página 447
     - "Preparación de programas COBOL" en la página 448
     - "Preparación de programas pTAL" en la página 449

Ejemplos de IBM WebSphere MQ para Windows

Este tema muestra las técnicas demostradas por los programas de ejemplo para IBM WebSphere MQ para Windows.

Tabla 16 en la página 109 La tabla lista qué archivos fuente C y COBOL se proporcionan, y si se incluye un ejecutable de servidor o cliente.

Tabla 16. Programas de ejemplo de IBM WebSphere MQ para Windows que muestran el uso de 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

Tabla 16. Programas de ejemplo de IBM WebSphere MQ para Windows que muestran el uso de MQI (C y COBOL) (continuación)

Técnica	C (fuente)	COBOL (fuente)	Servidor (ejecutable C)	Cliente (ejecutable C)
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
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

Tabla 16. Programas de ejemplo de IBM WebSphere MQ para Windows que muestran el uso de MQI (C y COBOL) (continuación)

Técnica	**C (fuente)**	**COBOL (fuente)**	**Servidor (ejecutable C)**	**Cliente (ejecutable C)**
WebSphere MQ (coordinación de gestores de bases de datos compatibles con XA) que acceden a una única base de datos utilizando SQL	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	no hay ejemplo	no hay ejemplo
WebSphere MQ (coordinación de gestores de bases de datos compatibles con XA) que acceden a dos bases de datos utilizando 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 112)	no hay ejemplo	amqsdlq	no hay ejemplo
Desde un cliente MQI de WebSphere MQ , colocar un mensaje	no hay ejemplo	no hay ejemplo	no hay ejemplo	amqsputc
Desde un cliente MQI de WebSphere MQ , obteniendo un mensaje	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

Tabla 16. Programas de ejemplo de IBM WebSphere MQ para Windows que muestran el uso de MQI (C y COBOL) (continuación)

Técnica	C (fuente)	COBOL (fuente)	Servidor (ejecutable C)	Cliente (ejecutable C)
Utilización de consumidores de mensajes para consumir mensajes de varias colas asíncronamente	amqscbf0	no hay ejemplo	amqscbf	amqscbfc
Especificación de información de conexión SSL/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.

Ejemplos de Visual Basic para IBM WebSphere MQ para Windows

Este tema muestra las técnicas demostradas por los programas de ejemplo de Visual Basic para IBM WebSphere MQ para Windows.

La Tabla 17 en la página 112 muestra las técnicas demostradas por los programas de ejemplo de IBM WebSphere MQ para 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 WebSphere MQ. Para descubrir cómo cambiar los proyectos de ejemplo para que funcionen con clientes de IBM WebSphere MQ, consulte “Preparación de programas de Visual Basic en Windows” en la página 473.

Tabla 17. Programas de ejemplo de IBM WebSphere MQ para Windows que muestran el uso de MQI (Visual Basic)

Técnica	Nombre de archivo de proyecto
Colocación de mensajes mediante la llamada MQPUT	amqsputb.vbp
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

Preparación y ejecución de los programas de ejemplo

Configure el gestor de colas para que acepte de forma segura las solicitudes de conexión entrantes de las aplicaciones que se ejecutan en modalidad de cliente.

Antes de empezar

Asegúrese de que el gestor de colas ya existe y se ha iniciado. Determine si los registros de autenticación de canal ya están habilitados de la siguiente manera:

```
DISPLAY QMGR CHLAUTH
```

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 el gestor de colas no utiliza registros de autenticación de canal, el paso “4” en la página 113 se puede sustituir por un método de autenticación alternativo (por ejemplo, una salida de seguridad) que establece el MCAUSER en el *id-usuario-no-privilegiado* que obtendrá en el paso “1” en la página 113.

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 es el nombre del gestor de colas
   nnnn es el número de puerto elegido

   a) Para sistemas UNIX y Windows :

   ```
   runmqlsr -t tcp -m qmgr -p nnnn
   ```

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')
   ```

   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 113
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')
   ```

queue-name es el nombre de la cola.

6. Otorgue acceso para conectar con el gestor de colas y consultarlo:
   a) Para UNIX yWindows emiten los mandatos MQSC:

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

   non-privileged-user-id es el ID de usuario que ha obtenido en el paso "1" en la página 113
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:
   a) Para UNIX y Windows emiten los mandatos MQSC:

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

   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 113
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:
   a) Para UNIX y Windows emiten los mandatos MQSC:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL('non-privileged-user-id') AUTHADD(PUB, SUB)
```

   non-privileged-user-id es el ID de usuario que ha obtenido en el paso "1" en la página 113
   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.

Tareas relacionadas

Otorgar acceso a un objeto WebSphere MQ en sistemas UNIX o Linux y Windows

Referencia relacionada

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

Preparación y ejecución de programas de ejemplo en sistemas UNIX

Tabla 18. Dónde encontrar los ejemplos para WebSphere MQ en sistemas UNIX 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 WebSphere MQ.	

Los archivos de ejemplo de WebSphere MQ en sistemas UNIX and Linux se encuentran en los directorios listados en Tabla 18 en la página 114 si se han utilizado los valores predeterminados durante la instalación. 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. Para obtener información sobre cómo hacerlo, consulte “Ejecución de los programas de ejemplo” en la página 115.

Preparación y ejecución de programas de ejemplo en sistemas Windows

Tabla 19. Dónde encontrar los ejemplos de WebSphere MQ para 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 WebSphere MQ.

Nota:

1. Hay disponibles versiones de 64 bits de algunos ejemplos de archivos ejecutables C.

Los archivos de ejemplo de WebSphere MQ para Windows están en los directorios listados en Tabla 19 en la página 115 si se han utilizado los valores predeterminados durante la instalación; la unidad de instalación toma el valor predeterminado < c: >. Para ejecutar los ejemplos, utilice las versiones ejecutables proporcionadas o compile las versiones de origen como lo haría con cualquier otra aplicación de WebSphere MQ para Windows . Si desea más información sobre cómo hacerlo, consulte “Ejecución de los programas de ejemplo” en la página 115.

Ejecución de los programas de ejemplo

Considere la posibilidad de utilizar este tema al ejecutar programas de ejemplo en distintas plataformas.

Para poder ejecutar cualquiera de los programas de ejemplo, cree un gestor de colas y configure las definiciones predeterminadas. Esto se explica en Administración.

En plataformas Windows, UNIX y Linux

Los ejemplos necesitan un conjunto de colas con las que trabajar. Utilice sus propias colas o ejecute el archivo MQSC de ejemplo amqscos0.tst para crear un conjunto.

Para hacerlo en sistemas UNIX and Linux , especifique:

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

Compruebe el archivo `sampobj.out` para asegurarse de que no se hayan producido errores.

Para hacerlo 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.

Ahora puede ejecutar las aplicaciones de ejemplo. Especifique el nombre de la aplicación de ejemplo seguido de 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`.

Consulte la descripción de los ejemplos individuales para obtener información sobre los parámetros que espera cada uno de ellos.

Longitud del nombre de cola

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.

Ejemplos de inquire, Set y Echo

Para los ejemplos Inquire, Set y Echo, las definiciones de ejemplo desencadenan las versiones C de estos ejemplos.

Si desea las versiones de COBOL, debe cambiar las definiciones de proceso:

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

En Windows, los sistemas UNIX and Linux lo hacen editando el archivo `amqscos0.tst` y cambiando los nombres de archivo ejecutable C por los nombres de archivo ejecutable COBOL antes de utilizar el mandato **runmqsc** , tal como se muestra anteriormente.

El programa de ejemplo de salida de API

La salida de API de ejemplo genera un rastreo MQI para 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 la página 396.

Fuente
amqsaxe0.c

Binario
amqsaxe

Configuración para la salida de ejemplo

1. Añada lo siguiente al 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 WebSphere 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 WebSphere MQ está instalado.

2. Establecer la variable de entorno

```
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 en código fuente C y como binarios de cliente y servidor ejecutables en las plataformas Windows y UNIX and Linux.

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: El *Tipo de reconexión* solo es válido en programas cliente.

Ejemplo

El ejemplo muestra amqscbf ejecutando como programa de servidor que lee un mensaje de QL1 y luego se para.

Utilice WebSphere 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 de consumo asíncrono, no es necesario realizar ningún sondeo, y la gestión de hebras y almacenamiento la realiza WebSphere 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 “Características demostradas en los programas de ejemplo” en la página 100 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 “Características demostradas en los programas de ejemplo” en la página 100 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.

Sistemas UNIX, Linux y Windows

Considere la posibilidad de utilizar este tema cuando aprende a examinar programas de ejemplo en sistemas UNIX, Linux y 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.

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 “Características demostradas en los programas de ejemplo” en la página 100 para los nombres de estos programas.

Este programa toma estos parámetros:

1. El nombre de la cola de origen
2. Nombre del gestor de colas
3. Un parámetro opcional para las propiedades.

Los dos primeros parámetros de entrada para este programa son obligatorios. Por ejemplo, inicie el programa de una de las maneras 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)

Los valores permitidos para el parámetro de propiedades son:

Valor	Comportamiento
0	Comportamiento predeterminado, como era para V6. Las propiedades que se entregan a la aplicación dependen del atributo de cola *PropertyControl* del que se recupera el mensaje.

Valor	Comportamiento

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 de la versión 6; 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.

Consulte Administración para obtener un ejemplo de la salida de este programa de utilidad.

El ejemplo de transacción CICS

Se proporciona un programa de transacción CICS de ejemplo, denominado amqscic0.ccs para el código fuente y amqscic0 para la versión ejecutable. Puede crear transacciones utilizando los recursos CICS estándar.

Consulte “Creación de una aplicación IBM WebSphere MQ” en la página 438 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]] [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

SvrconnChannelName
Nombre del canal de conexión del servidor

QMgrName
Nombre del gestor de colas de destino

Si no especifica el nombre de conexión TCP/IP, MQCONNX se emite con *ClientConnPtr* establecido en NULL. 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. 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 “Características demostradas en los programas de ejemplo” en la página 100 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 muestran cómo WebSphere MQ puede coordinar tanto las actualizaciones de WebSphere MQ como 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 única base de datos dentro de una unidad de trabajo de WebSphere 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 dentro de una unidad de trabajo de WebSphere MQ , que muestra cómo se puede acceder a varias bases de datos. Estos ejemplos se proporcionan para mostrar el uso de las llamadas MQBEGIN, SQL mixto y WebSphere MQ , y dónde y cuándo conectarse a una base de datos.

Figura 18 en la página 124 muestra cómo se utilizan los ejemplos proporcionados para actualizar bases de datos:

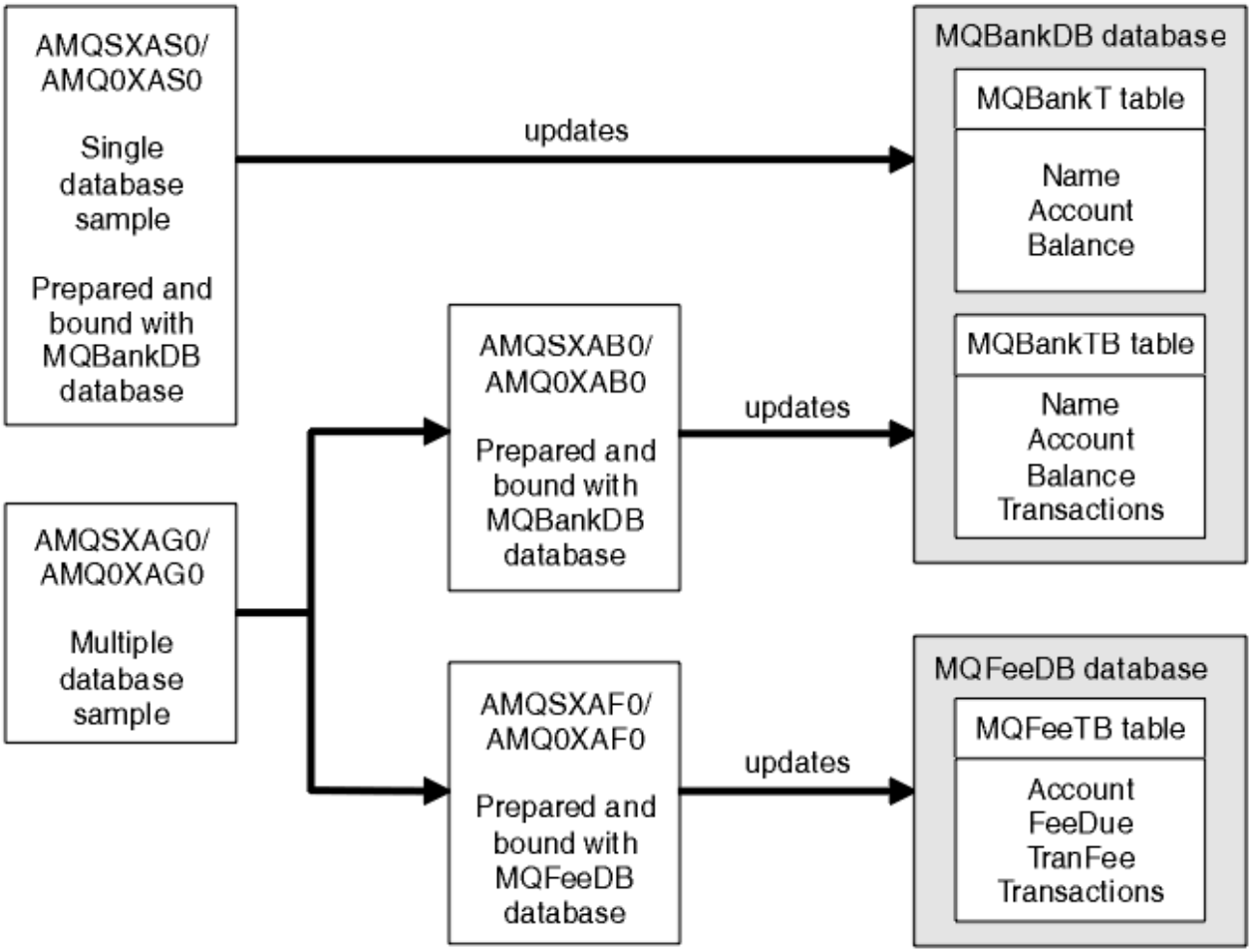


Figura 18. Ejemplos de coordinación de bases de datos

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 sentencias CONNECT de SQL y la conexión con la base de datos la realiza implícitamente WebSphere 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 es posible que tenga que modificarlos para que funcionen con otros productos de base de datos.

La comprobación de errores de SQL utiliza rutinas en UTIL.C y CHECKERR.CBL proporcionado 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 adecuado 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 los símbolos *<DB2TOP>* y *MQ_INSTALLATION_PATH*. *<DB2TOP>* representa el directorio de instalación para el producto DB2 . *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ .

- En AIX, la vía de acceso del directorio es:

```
/usr/lpp/db2_05_00
```

- En HP-UX y Solaris, la vía de acceso del directorio es:

```
/opt/IBMdb2/V5.0
```

- En sistemas Windows , la vía de acceso del directorio depende de la vía de acceso elegida al instalar el producto. Si selecciona los valores predeterminados, la vía de acceso es:

```
c:\sqllib
```

Nota: Antes de emitir el mandato de enlace en sistemas Windows , asegúrese de que la variable de entorno LIB contiene vías de acceso a las bibliotecas DB2 y WebSphere MQ .

Copie los archivos siguientes en un directorio temporal:

- El archivo `amqsxag0.c` de la instalación de WebSphere MQ

  Nota: Este archivo se puede encontrar en los directorios siguientes:

  – En sistemas UNIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – En sistemas Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Los archivos `.c` que ha obtenido precompilando los archivos de origen de `.sqc` , `amqsxas0.sqc`, `amqsxaf0.sqc` y `amqsxab0.sqc`
- Los archivos `util.c` y `util.h` de la 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

```
xlc_r -IMQ_INSTALLATION_PATH/inc -I
<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- HP-UX

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

- Solaris

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

- Sistemas Windows

```
cl /c /IMQ_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

```
xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib
 -lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- HP-UX Revisión 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread -lcl
 /lib/crt0.o util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris

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

- Sistemas Windows

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
 /out:amqsxag0.exe
```

Cree el archivo ejecutable amqsxas0 utilizando los siguientes mandatos de compilación y enlace para la plataforma que está utilizando:

- AIX

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

- HP-UX Revisión 11i

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

- Solaris

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

- Sistemas Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Información adicional

Si está trabajando en AIX o HP-UX y desea acceder a Oracle, utilice el compilador xlc_r y enlace a 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 conseguirlo consulte el apartado "Escenario 1: El gestor de colas realiza la coordinación" en la página 44.

Los títulos siguientes proporcionan información sobre cómo ejecutar ejemplos en C y COBOL:

- "Ejemplos en C" en la página 129
- "Ejemplos en COBOL" en la página 130

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 y el nombre de la versión del ejecutable es amqsdlq. Si desea un manejador de cola de mensajes no entregados diferente a RUNMQDLQ, el código fuente 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
Establézcala 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 nombre del archivo proporcionado es 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 mensajes en stdout, ***no*** en el archivo de registro de errores de WebSphere MQ .

En la sección Administración o en la *Guía de gestión del sistema* de su plataforma, se describe el funcionamiento del manejador de la cola de mensajes no entregados y cómo ejecutarlo.

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 valores para *MsgId* y *CorrelId*, y estos alteran temporalmente 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. Los registros de objeto (MQOR) requieren 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 utilizan los mismos registros de respuesta para cada llamada que indica *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 "Características demostradas en los programas de ejemplo" en la página 100 para los nombres de estos programas.

Los programas están pensados para ejecutarse como programas desencadenados.

En sistemas UNIX, Linux y Windows , su única entrada es una estructura MQTMC2 (mensaje desencadenante) que contiene el nombre de una cola de destino y el gestor de colas. La versión en COBOL utiliza el gestor de colas predeterminado.

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. (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, 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.

Programas de ejemplo de obtención

Los programas de ejemplo de obtención obtienen los mensajes de una cola utilizando la llamada MQGET.

Consulte "Características demostradas en los programas de ejemplo" en la página 100 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

Cada uno de estos programas toma dos parámetros:

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 una variable de entorno o el archivo de definición de canal 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`.

Si omite `qmanagername`, los programas asumen el valor predeterminado o, en el caso del cliente MQI, el gestor de colas identificado por una variable de entorno o el archivo de definición de canal de cliente.

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 136.

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 *queueName* **[***qMgrNombre***]**

- **amqsphac** es una aplicación IBM WebSphere 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 *queueName* **[***qMgrNombre***]**

- **amqsghac** es una aplicación IBM WebSphere 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 *sourceQueueNombre* **-t** *targetQueueNombre* **[-m** *qMgrNombre***] [-w** *waitInterval***]**

- **amqsmhac** es una aplicación IBM WebSphere 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 *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* **(0|1|2)**

- **amqsfhac** es una aplicación IBM WebSphere MQ MQI client . Comprueba que un gestor de colas de varias instancias de IBM WebSphere 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 conectarse a *QueueManagerNombre*. Se reconecta automáticamente cuando el gestor de colas falla.
- Coloca *InTransactionCount***RepeatCount* mensajes persistentes en *QueueName* durante el cual hace que el gestor de colas realice la migración tras error 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.
- Los mensajes de *InTransactionCount* se colocan dentro de cada transacción. La transacción se repite *RepeatCount* número de veces. 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 coloca mensajes en *SideQueueNombre*. Utiliza *SideQueueNombre* para comprobar si todos los mensajes se han confirmado o retrotraído de *QueueName* correctamente. Si detecta una incoherencia, escribe un mensaje de error.
- Varíe la cantidad de rastreo de salida de **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. Consulte Verificación de una instalación 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 19 en la página 135.

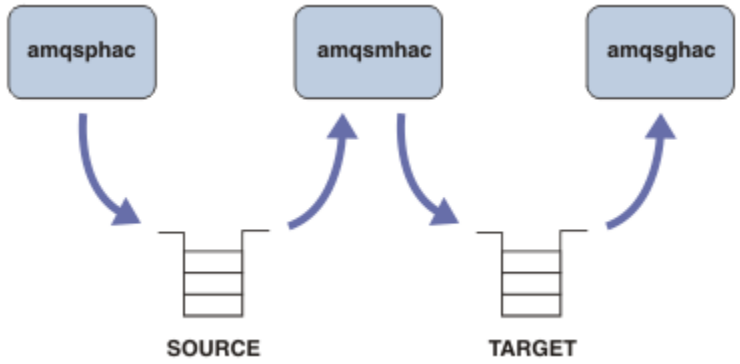


Figura 19. Ejemplos de cliente reconectable

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** establecido; por ejemplo, en Windows, escriba **start** tres veces en el indicador de mandatos anterior iniciando cada programa en una de las ventanas. Consulte el paso “5” en la página 136 en “Configurar y controlar el gestor de colas” en la página 136.
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 los ejemplos **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
WebSphere MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Recuerde el directorio de datos para establecer la variable **MQCHLLIB** más adelante.

2. Inicie el gestor de colas.

```
C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

3. 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: WebSphere MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: WebSphere MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: WebSphere MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: WebSphere MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: WebSphere MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start WebSphere MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start WebSphere MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. 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 136, 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.
WebSphere MQ queue manager 'QM1' ending.
WebSphere MQ queue manager 'QM1' ended.

C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
```

```
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
<Message 3>
message <Message 4>
message <Message 5>
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message <Message 3>
message <Message 4>
message <Message 5>
```

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 “Características demostradas en los programas de ejemplo” en la página 100 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 sistemas IBM i, Windows, UNIX and Linux . 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. 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:

- Para los sistemas UNIX, Linux y Windows , inicie el programa **runmqtrm** en una sesión y, a continuación, inicie el programa `amqsreq` en otra.

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.

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.

Para cada mensaje de solicitud eliminado de la cola de solicitudes, el programa lee el nombre de la cola (que llamaremos la *cola de destino*) contenida en los datos y abre dicha cola utilizando la llamada MQOPEN con la opción MQOO_INQ. A continuación, 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:\Program Files\IBM\WebSphere MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name <MyProp> value <MyValue>
message text <Hello world!>
Sample AMQSIQMA end
```

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 WebSphere MQ.

Hay tres programas de ejemplo de lenguaje C que ilustran cómo programar en la interfaz de publicación/suscripción de WebSphere MQ . Hay algunos ejemplos de C que utilizan interfaces más antiguas, y hay

ejemplos de Java. Los ejemplos de Java utilizan la interfaz de publicación/suscripción WebSphere MQ en com.ibm.mq.jar y la interfaz de publicación/suscripción 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 "Preparación y ejecución de los programas de ejemplo" en la página 112 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 entre en esta ventana a partir de ahora se publica en `mytopic`.

Abra otra ventana de mandatos 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 20 en la página 139 se muestran seis tipos de ejemplo.

Tabla 20. 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.

Tabla 20. Categorías de programas de ejemplo de publicación/suscripción en C legados (continuación)		
Categoría	**Programas**	**Comentarios**
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` debe iniciarse antes de `amqsgama`
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` debe iniciarse 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.

Java

El ejemplo Java `MQPubSubApiSample.java` combina el publicador y los suscriptores en un único 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 "Preparación y ejecución de los programas de ejemplo" en la página 112 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 WebSphere MQ Explorer Eclipse que ya tiene configurado un entorno de trabajo de programación Java.

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 140 muestran cómo ejecutar el ejemplo en el espacio de trabajo de Eclipse.

Ejecución del ejemplo Java MQPubSubApiSample

Cómo ejecutar MQPubSubApiSample utilizando las herramientas de desarrollo Java de 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 “Preparación y ejecución de los programas de ejemplo” en la página 112 antes de ejecutar como cliente.

Acerca de esta tarea

El programa de ejemplo de publicación/suscripción Java es un programa Java de cliente MQI de WebSphere MQ . 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 que se adapte a las distintas configuraciones de WebSphere MQ . El ejemplo se ilustra cuando se ejecuta en Windows. Las vías de acceso de archivo serán diferentes en otras plataformas.

Procedimiento

1. Importar los programas de ejemplo Java
   a) En el entorno de trabajo, pulse **Ventana** > **Abrir perspectiva** > **Otros** > **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 WebSphere 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 WebSphere MQ .
   k) En el panel **Importar**, Figura 20 en la página 142, pulse `samples` (no marque el recuadro de selección).
   l) Seleccione `MQPubSubApiSample.java`. El campo **`Into folder`** debe contener `MQ Java Samples/src`. Pulse **Finalizar**.

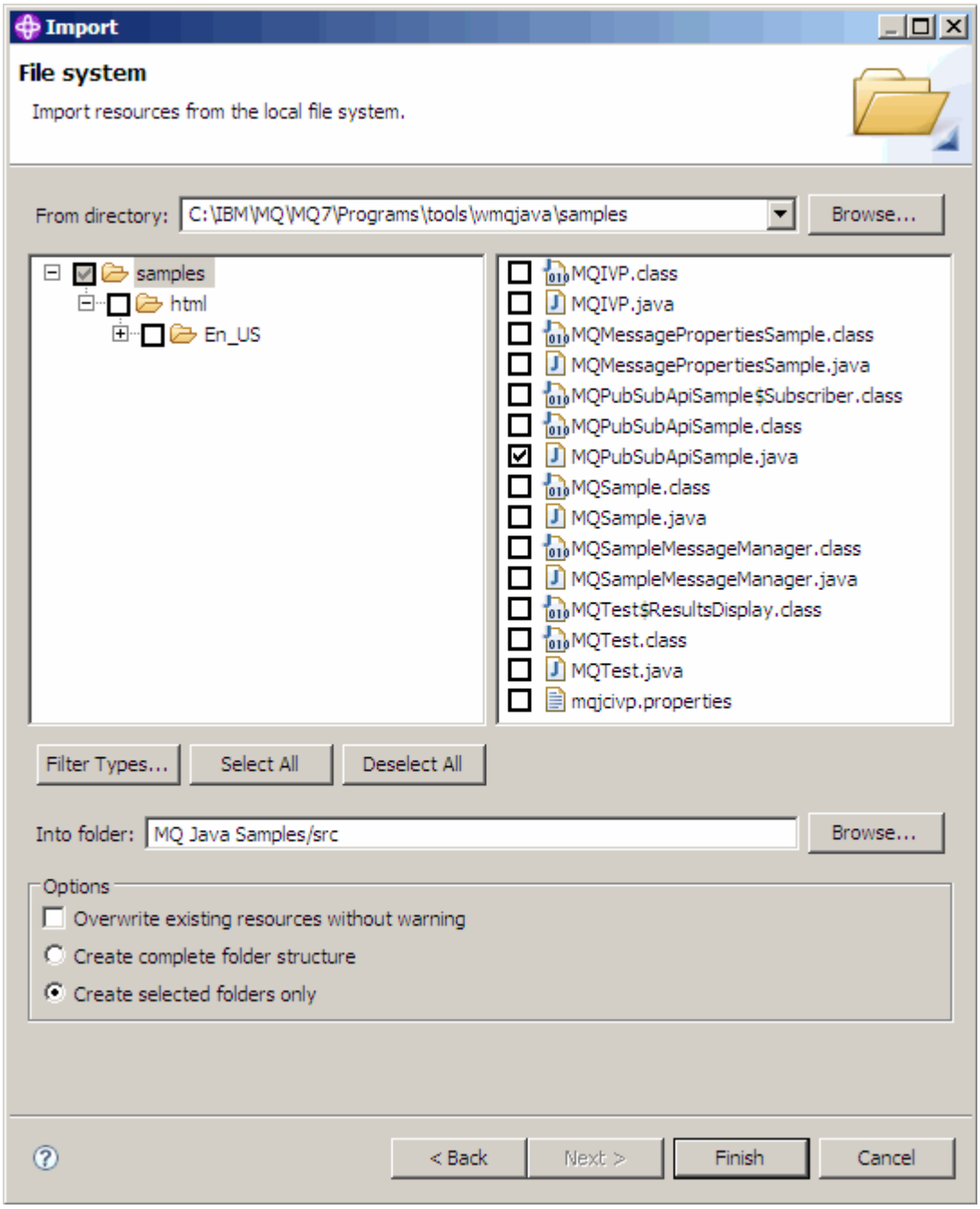


Figura 20. Importación de sistema de archivos

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 el botón derecho del ratón sobre **MQPubSubApiSample.java Ejecutar como** > **1. Aplicación Java**
   - 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 JVM o proporcionando un valor predeterminado para la propiedad Sistema editando el código fuente.

  - topicObject
  - queueManagerName
  - subscriberCount

  Estos atributos solo son modificables editando el código fuente en el constructor.

  - hostname
  - puerto
  - canal

  Para establecer las propiedades del sistema, 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 con el botón derecho del ratón en **Ejecutar** > **Abrir diálogo Ejecutar**. Efectúe una doble pulsación en Aplicación Java en **Crear, gestionar y ejecutar aplicaciones** y pulse la pestaña **(x) = Argumentos** .
c. En el panel **Argumentos de VM:** , escriba `-D` y pegue el nombre System.property , `com.ibm.mq.pubSubSample.queueManagerName`, seguido de `=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:
   - En los sistemas UNIX 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 WebSphere MQ . En Windows, establezca los atributos equivalentes en el registro.
2. Asegúrese de que el módulo sea accesible para WebSphere 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:

- En sistemas UNIX and Linux , asegúrese de que el directorio `/var/mqm/trace` existe y exporte la siguiente variable de entorno:

```
export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
```

- En Windows, asegúrese de que el directorio `C:\temp` existe y establezca la siguiente variable de entorno:

```
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 "Características demostradas en los programas de ejemplo" en la página 100 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

Cada uno de estos programas toma 2 parámetros:

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 una variable de entorno o el archivo de definición de canal de cliente. Para ejecutar estos programas, especifique una de las opciones siguientes:

- `amqsput myqueue qmanagername`
- `amqsputc 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`.

Ejecución del ejemplo amq0put

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.

Programas de ejemplo de mensaje de referencia

Los ejemplos de mensajes de referencia permiten transferir un objeto grande de un nodo a otro (normalmente en sistemas diferentes) sin necesidad de que el objeto se almacene en colas de WebSphere MQ en los nodos de origen o 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 bases de datos, o si desea realizar comprobaciones de seguridad, defina su propia salida, basándose en nuestro ejemplo, amqsxrm. En las secciones siguientes se describen los programas de ejemplo de mensajes de referencia.

La versión del programa de ejemplo de salida de mensajes de referencia que se debe utilizar depende de la plataforma en la que se ejecuta el canal. En todas las plataformas, utilice amqsxrma en el extremo emisor. Utilice amqsxrma en el extremo receptor si el receptor se ejecuta en cualquier producto WebSphere MQ excepto WebSphere MQ para IBM i.

Ejecución de los ejemplos de mensajes de referencia

Utilice esta información para aprender a ejecutar programas de ejemplo de mensajes de referencia.

Los ejemplos de Mensaje de referencia se ejecutan de la forma siguiente:

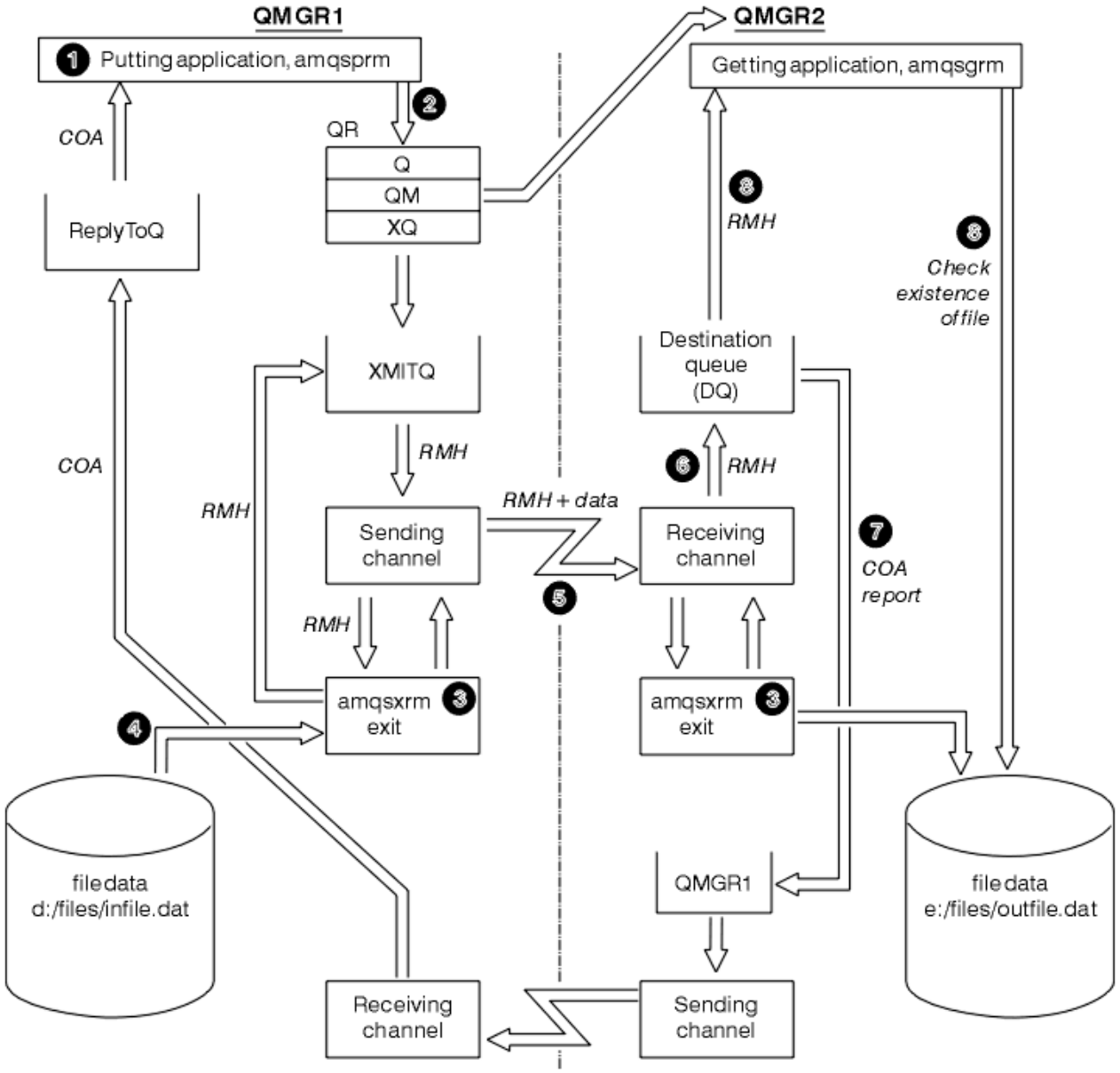


Figura 21. Ejecución de los ejemplos de mensajes de referencia

1. Configure el entorno para iniciar los escuchas, los canales y los supervisores de desencadenante, y defina los canales y las colas.

   Para describir cómo configurar el ejemplo de mensaje de referencia, esto hace referencia a la máquina emisora como MACHINE1 con un gestor de colas denominado QMGR1 y a la máquina receptora como MACHINE2 con un gestor de colas denominado 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) 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 donde esté instalado el producto WebSphere MQ .

Las definiciones MQSC han utilizado un estilo AIX para definir las salidas. 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
```

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)
```

2. Una vez creados los objetos de WebSphere 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 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 cola

-g Nombre del gestor de colas donde existe la cola, definida en el parámetro -q. Toma como valor predeterminado 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
```

Nota: Para las plataformas UNIX 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 utilizado 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. Esto se define en WebSphere MQ para Windows de la forma siguiente:

```
msgexit('pathname\amqsxrm.dll(MsgExit)')
```

Esto se define en WebSphere MQ para AIX, WebSphere MQ para HP-UXy WebSphere MQ para Solaris 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 presupone que el programa está en la vía de acceso especificada en el archivo `qm.ini` (o, en WebSphere MQ para 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. En WebSphere MQ para AIX, WebSphere MQ para HP-UXy WebSphere MQ para Solaris, 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 WebSphere 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 que se va 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 ejemplo, el ejemplo que se proporciona con WebSphere MQ crea un mensaje de referencia para enviar el archivo `d:\x\file.in` de `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: En las plataformas UNIX and Linux, utilice 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 -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 o el *CodedCharacterSetId* (CCSID) del mensaje de referencia es diferente de la del gestor de colas, el mensaje de referencia se convierte a la codificación local y al CCSID. 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

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Para compilar amqsxrma, utilice estos mandatos:

En AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

en HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -IMQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -LMQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

enLinux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

En Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm

 -lsocket
-lnsl -ldl
```

En Windows

WebSphere MQ ahora proporciona 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
```

Para obtener información general sobre cómo escribir y compilar salidas de canal, consulte "Cómo escribir programas de salida de canal" en la página 408

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 "Características demostradas en los programas de ejemplo" en la página 100 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.

Sistemas UNIX, Linux y Windows

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 22 en la página 153 ilustra cómo utilizar los ejemplos de solicitud y consulta juntos.

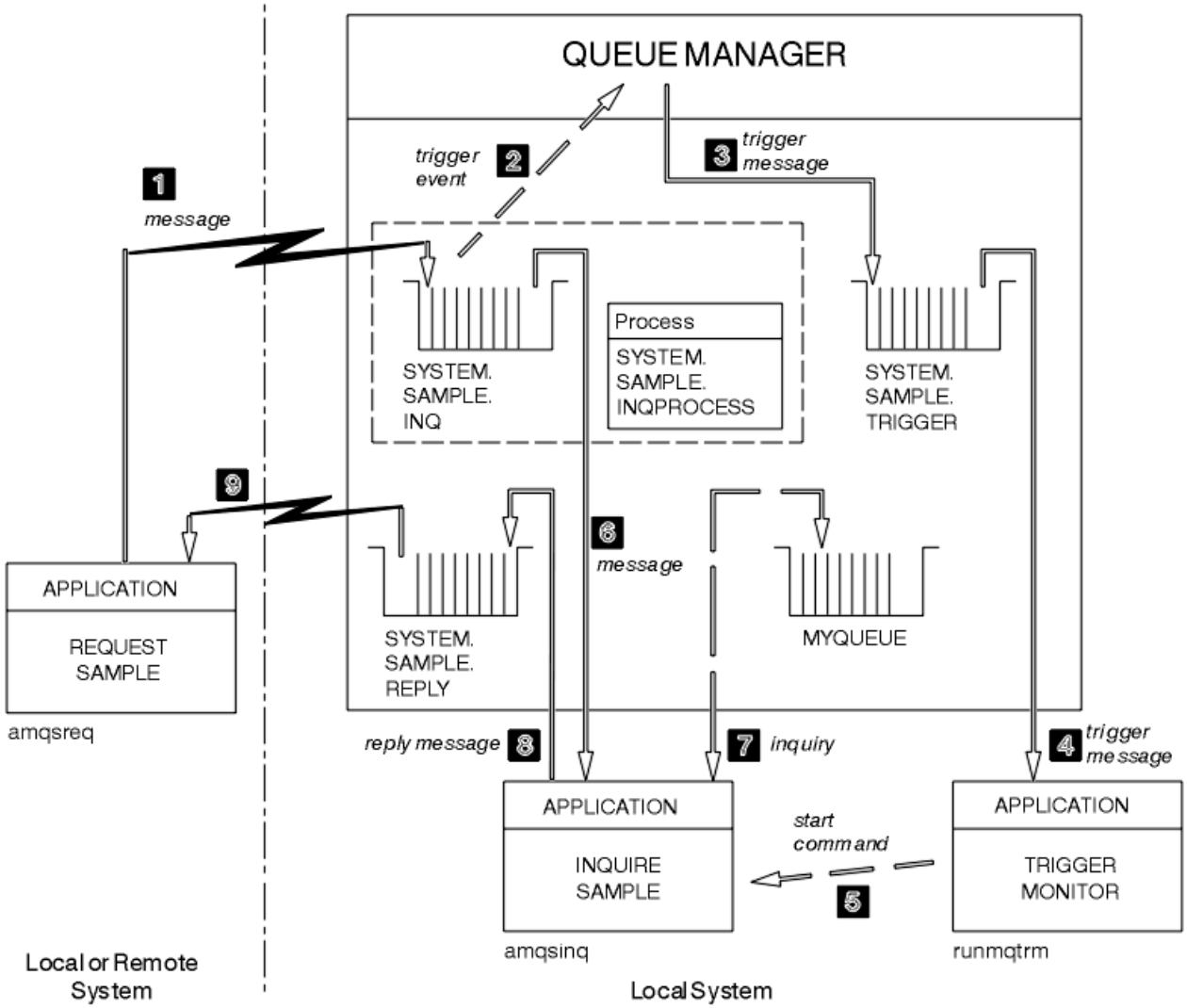


Figura 22. Ejemplos de solicitud y consulta que utilizan el desencadenamiento

En Figura 22 en la página 153, el ejemplo de solicitud coloca mensajes en la cola del servidor de destino, SYSTEM.SAMPLE.INQ, y el ejemplo de consulta 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 22 en la página 153 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 154.

Para obtener más información sobre el desencadenante, consulte "Inicio de aplicaciones de IBM WebSphere MQ utilizando desencadenantes" en la página 337.

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* es de 5 minutos en la versión COBOL, y de 1 minuto en la versión C, para la primera respuesta (para dar tiempo a que se desencadene una aplicación de servidor), y de 15 segundos para 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.

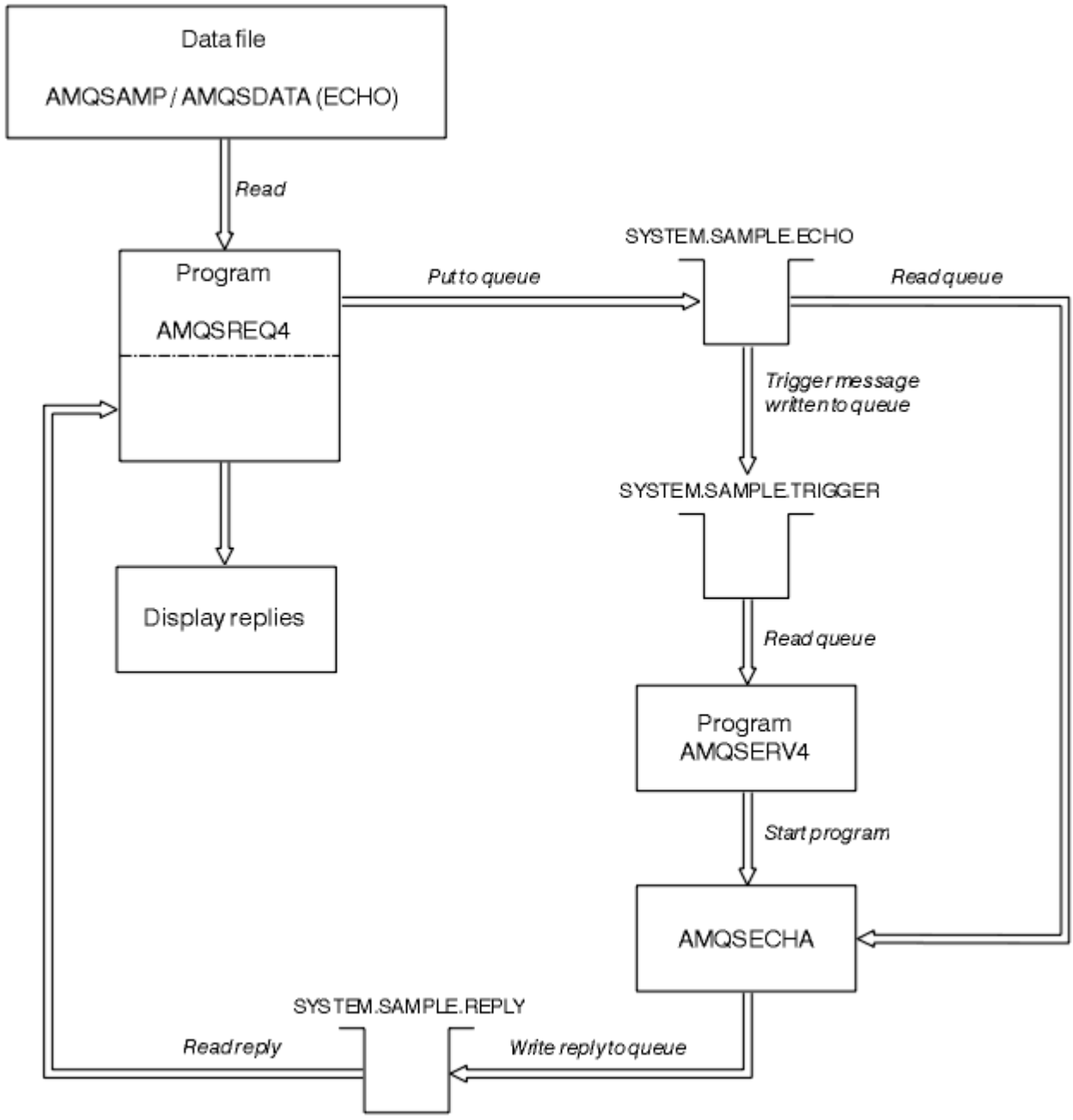


Figura 23. Diagrama de flujo del programa de ejemplo de cliente/servidor IBM i (eco)

Programas Set de ejemplo

Los programas de ejemplo Set 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 “Características demostradas en los programas de ejemplo” en la página 100 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:

- Para los sistemas UNIX, Linux y Windows , inicie el programa **runmqtrm** en una sesión y, a continuación, inicie el programa amqsreq en otra.
- 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.

Para cada mensaje de solicitud eliminado de la cola de solicitudes, el programa lee el nombre de la cola (a la que llamaremos la *cola de destino*) contenida 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_PROGRITED.

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 SSL/TLS

AMQSSLC es un programa C de ejemplo que muestra cómo utilizar las estructuras MQCNO y MQSCO para proporcionar información de conexión de cliente SSL/TLS en la llamada MQCONNX. Esto permite que una

aplicación MQI de cliente proporcione la definición de su canal de conexión de cliente y valores SSL/TLS en tiempo de ejecución sin una tabla de definiciones de canal 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. No obstante, también se puede enlazar como una aplicación MQI habitual. A continuación, simplemente se conecta a un gestor de colas y omite la información de conexión del cliente.

AMQSSLC 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 SSL/TLS:

-k KeyReposStem
El nombre de la raíz del archivo del repositorio de claves. Este nombre es la vía de acceso completa del archivo sin el sufijo .kdb. Por ejemplo:

```
/home/user/client
C:\User\client
```

-s CipherSpec
La serie CipherSpec del canal SSL/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 de 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.

Parámetro de revocación de certificados OCSP:

-o URL
El URL de respondedor OCSP

Ejecución del programa de ejemplo SSL/TLS

Para ejecutar el programa de ejemplo SSL/TLS, primero debe configurar el entorno SSL o 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 sistemas UNIX, Linux, and Windows.
3. Cree un repositorio de claves para el cliente. Llámelo *clientkey.kdb*.
4. Cree un certificado personal para el gestor de colas. Para obtener más información, consulte Creación de un certificado personal autofirmado en sistemas UNIX, 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 sistemas UNIX, Linux y Windows y Adición de un certificado CA (o la parte pública de un certificado autofirmado) en un repositorio de claves, en sistemas UNIX, Linux o 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(NULL_SHA)
   ```

   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:

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "c:\Program Files\IBM\WebSphere MQ\clientkey" -s NULL_SHA
    -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 SSL CipherSpec NULL_SHA
Using SSL key repository stem c:\Program Files\IBM\WebSphere MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

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 la lista de códigos de razón, consulte Códigos de razón del 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 “Características demostradas en los programas de ejemplo” en la página 100 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.

Para los mensajes desencadenantes válidos, el ejemplo de desencadenamiento copia datos de estos campos: *ApplicId*, *EnvrData*, *Versio*n y *ApplType*. Los dos últimos de estos campos son numéricos, por lo que el programa crea sustituciones de caracteres para utilizar en una estructura MQTMC2 para sistemas UNIX, Linuxy Windows .

El ejemplo de desencadenamiento emite un mandato de inicio para la aplicación especificada en el campo *ApplicId* del mensaje desencadenante y pasa una estructura MQTMC2 o MQTMC (una versión de caracteres del mensaje desencadenante). En los sistemas UNIX, Linux y Windows , el campo *EnvData* se utiliza como una extensión de la serie de mandato de invocación.

Por último, el programa cierra la cola de inicio.

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** .

Diseño del servidor desencadenante

El diseño del servidor desencadenante es similar al del supervisor desencadenante, salvo que el servidor desencadenante:

- Permite aplicaciones MQAT_CICS así como MQAT_OS400
- Para aplicaciones CICS , sustituya *EnvData*, por ejemplo, para especificar la región CICS , desde el mensaje desencadenante en el mandato STRCICSUSR
- Abre la cola de inicio para la entrada compartida, de modo que muchos servidores desencadenantes se pueden ejecutar al mismo tiempo

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.

ejemplos de TUXEDO

Obtenga información sobre los programas de ejemplo Put y Get de TUXEDO y sobre cómo crear el entorno de servidor en TUXEDO.

Antes de ejecutar estos ejemplos, debe crear el entorno de servidor.

Nota: En este tema, se utiliza el carácter de barra invertida (\) para dividir los mandatos largos en más de una línea. No especifique este carácter. Entre cada mandato como una sola línea.

Creación del entorno de servidor

Información sobre cómo crear el entorno de servidor para WebSphere MQ para distintas plataformas.

Se asume que existe un entorno TUXEDO operativo.

Creación del entorno de servidor para WebSphere MQ para AIX (32 bits)

1. Cree un directorio (por ejemplo, < APPDIR>) en el que se cree el entorno de servidor y ejecute todos los mandatos de este 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 que está instalado WebSphere 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 lo 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
```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para AIX (64 bits)

1. Cree un directorio (por ejemplo, < APPDIR>) en el que se cree el entorno de servidor y ejecute todos los mandatos de este 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 que está instalado WebSphere MQ installed.:

```
$ 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 lo 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
```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para Solaris (32 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

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 de TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. Añada lo siguiente al archivo udataobj/RM de TUXEDO.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a
```

4. Ejecute los comandos:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
```

```
              -l -ldl
       $ buildclient -o doputs  -f amqstxpx.c \
              -f MQ_INSTALLATION_PATH/lib/libmqm.so  \

       $ buildclient -o dogets  -f amqstxgx.c \
              -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. 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 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
```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para Solaris (64 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

1. Cree un directorio (por ejemplo, < APPDIR>) en el que se cree el entorno de servidor y ejecute todos los mandatos de este directorio.
2. Exporte las variables de entorno siguientes, donde TUXDIR es el directorio raíz de TUXEDO:

```
       $ export CFLAGS="-I /<APPDIR>"
       $ export FIELDTBLS=amqstxvx.flds
       $ export VIEWFILES=amqstxvx.V
       $ export SHLIB_PATH=$TUXDIR□lib:MQ_INSTALLATION_PATH□lib:□lib64
       $ export LD_LIBRARY_PATH=$TUXDIR□lib64:MQ_INSTALLATION_PATH□lib64:□lib64
```

3. Añada lo siguiente al archivo udataobj/RM de TUXEDO.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a
```

4. Ejecute los comandos:

```
       $ mkfldhdr    amqstxvx.flds
       $ viewc       amqstxvx.v
       $ buildtms    -o MQXA -r MQSeries_XA_RMI
       $ buildserver -o MQSERV1 -f amqstxsx.c \
              -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
              -r MQSeries_XA_RMI -s MPUT1:MPUT \
              -s MGET1:MGET \
              -v -bshm
              -l -ldl
```

```
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs  -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so  \

$ buildclient -o dogets  -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. 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 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
```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para HP-UX (32 bits)

Nota: El entorno de servidor TUXEDO de 32 bits sólo se puede compilar en la plataforma Itanium.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

1. Cree un directorio (por ejemplo, < APPDIR>) en el que se cree el entorno de servidor y ejecute todos los mandatos de este directorio.
2. Exporte las variables de entorno siguientes, donde TUXDIR es el directorio raíz de TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Añada lo siguiente al archivo TUXEDO udataobj/RM

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. Ejecute los comandos:

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

Después de ejecutar los mandatos mkfldhdr y viewc, el archivo de cabecera amqstxvx.h se crea en el directorio de la aplicación TUXEDO. Copie este archivo del directorio de aplicación TUXEDO en el directorio de inclusión TUXEDO y, a continuación, ejecute los mandatos siguientes.

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. 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
```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para HP-UX (64 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

1. Cree un directorio (por ejemplo, < APPDIR>) en el que se cree el entorno de servidor y ejecute todos los mandatos de este directorio.
2. Exporte las variables de entorno siguientes, donde TUXDIR es el directorio raíz de TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Añada lo siguiente al archivo TUXEDO udataobj/RM

   En la plataforma HP-UX IA64 (IPF):

   ```
   MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
   MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
   /opt/tuxedo/lib/libtux.sl
   ```

   Nota: Las bibliotecas de WebSphere MQ que se suministran en la plataforma HP-UX IA64 (IPF) tienen una extensión de nombre de archivo .so.

4. Ejecute los comandos:

   ```
   $ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
   $ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
   ```

   Después de ejecutar los mandatos mkfldhdr y viewc, el archivo de cabecera amqstxvx.h se crea en el directorio de la aplicación TUXEDO. Copie este archivo del directorio de aplicación TUXEDO en el directorio de inclusión TUXEDO y, a continuación, ejecute los mandatos siguientes.

   ```
   $ buildtms    -o MQXA -r MQSeries_XA_RMI
   ```

   En la plataforma HP-UX IA64 (IPF):

   ```
   $ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
          -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
          -r MQSeries_XA_RMI -s MPUT1:MPUT \
          -s MGET1:MGET \
          -v -bshm
   $ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
          -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
          -r MQSeries_XA_RMI -s MPUT2:MPUT \
          -s MGET2:MGET \
          -v -bshm
   $ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
          -f MQ_INSTALLATION_PATH/lib64/libmqm.so
   $ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
          -f MQ_INSTALLATION_PATH/lib64/libmqm.so
   ```

5. 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
   ```

Ahora puede utilizar los programas de doputs y dogets para colocar mensajes en una cola y recuperarlos de la misma respectivamente.

Creación del entorno de servidor para WebSphere MQ para Windows (32 bits)

Nota: Cambie los campos identificados por <> en lo siguiente, por las vías de acceso de directorio:

< MQMDIR>	la vía de acceso del directorio especificada cuando se instaló WebSphere MQ , por ejemplo `g:\Program Files\IBM\WebSphere MQ`
< DIR_TUX>	la vía de acceso de directorio especificada cuando se instaló TUXEDO, por ejemplo, `f:\tuxedo`
< DIR_AP>	la vía de acceso del directorio que se utilizará para la aplicación de ejemplo, por ejemplo `f:\tuxedo\apps\mqapp`

Para crear el entorno de servidor y los ejemplos:

1. Cree un directorio de aplicación en el que compilar la aplicación de ejemplo, por ejemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie los siguientes archivos de ejemplo del directorio de ejemplo WebSphere 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 24 en la página 168) 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
   ```

   donde < MQMDIR > se sustituye tal como se muestra en el ejemplo anterior. Aunque se muestra aquí como dos líneas, la nueva entrada debe ser una 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 mandato:

   ```
   crdl -z <APPDIR>\TLOG
   ```

   donde <*APPDIR*> se sustituye.
8. Establezca el directorio actual en < 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:

   ```
   msvc amqstxmn.mak
   ```

   Seleccione **compilar** para compilar todos los programas de ejemplo.

```
*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 24. Ejemplo de archivo ubbstxcn.cfg para WebSphere MQ para Windows

Nota: Cambie los nombres de directorio y las vías de acceso de directorio para que coincidan con la instalación. Cambie también el nombre del gestor de colas MYQUEUEMANAGER por el nombre del gestor de colas al que desea conectarse. Otra información que necesita añadir se identifica mediante caracteres <> .

El archivo ubbconfig de ejemplo para WebSphere MQ para Windows se lista en Figura 24 en la página 168. Se proporciona como ubbstxcn.cfg en el directorio de ejemplos de WebSphere MQ .

El archivo make de ejemplo (consulte Figura 25 en la página 169) proporcionado para WebSphere MQ para Windows se denomina ubbstxmn.maky se encuentra en el directorio de ejemplos de WebSphere 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 25. Archivo make TUXEDO de ejemplo para WebSphere MQ para Windows

Creación del entorno de servidor para WebSphere MQ para Windows (64 bits)

Nota: Cambie los campos identificados por <> en lo siguiente, por las vías de acceso de directorio:

< MQMDIR>	la vía de acceso del directorio especificada cuando se instaló WebSphere MQ , por ejemplo `g:\Program Files\IBM\WebSphere MQ`
< DIR_TUX>	la vía de acceso de directorio especificada cuando se instaló TUXEDO, por ejemplo, `f:\tuxedo`
< DIR_AP>	la vía de acceso del directorio que se utilizará para la aplicación de ejemplo, por ejemplo `f:\tuxedo\apps\mqapp`

Para crear el entorno de servidor y los ejemplos:

1. Cree un directorio de aplicación en el que compilar la aplicación de ejemplo, por ejemplo:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copie los siguientes archivos de ejemplo del directorio de ejemplo WebSphere 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 26 en la página 170) para añadir los detalles del nombre de máquina y gestor de colas con el que desee 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
   ```

donde < MQMDIR > se sustituye. Aunque se muestra aquí como dos líneas, la nueva entrada debe ser una 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 mandato:

```
crdl -z <APPDIR>\TLOG
```

donde <*APPDIR*> se sustituye como se muestra en el ejemplo anterior.

8. Establezca el directorio actual en < 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:

```
msvc amqstxmn.mak
```

Seleccione **compilar** para compilar todos los programas de ejemplo.

```
*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 26. Ejemplo de archivo ubbstxcn.cfg para WebSphere MQ para Windows

Nota: Cambie los nombres de directorio y las vías de acceso de directorio para que coincidan con la instalación. Cambie también el nombre del gestor de colas MYQUEUEMANAGER por el nombre del gestor

de colas al que desea conectarse. Otra información que necesita añadir se identifica mediante caracteres <> .

El archivo ubbconfig de ejemplo para WebSphere MQ para Windows se lista en Figura 26 en la página 170. Se proporciona como ubbstxcn.cfg en el directorio de ejemplos de WebSphere MQ .

El archivo make de ejemplo (consulte Figura 27 en la página 171) proporcionado para WebSphere MQ para Windows se denomina ubbstxmn.maky se encuentra en el directorio de ejemplos de WebSphere 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 27. Archivo make TUXEDO de ejemplo para WebSphere MQ para Windows

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: Debe 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.

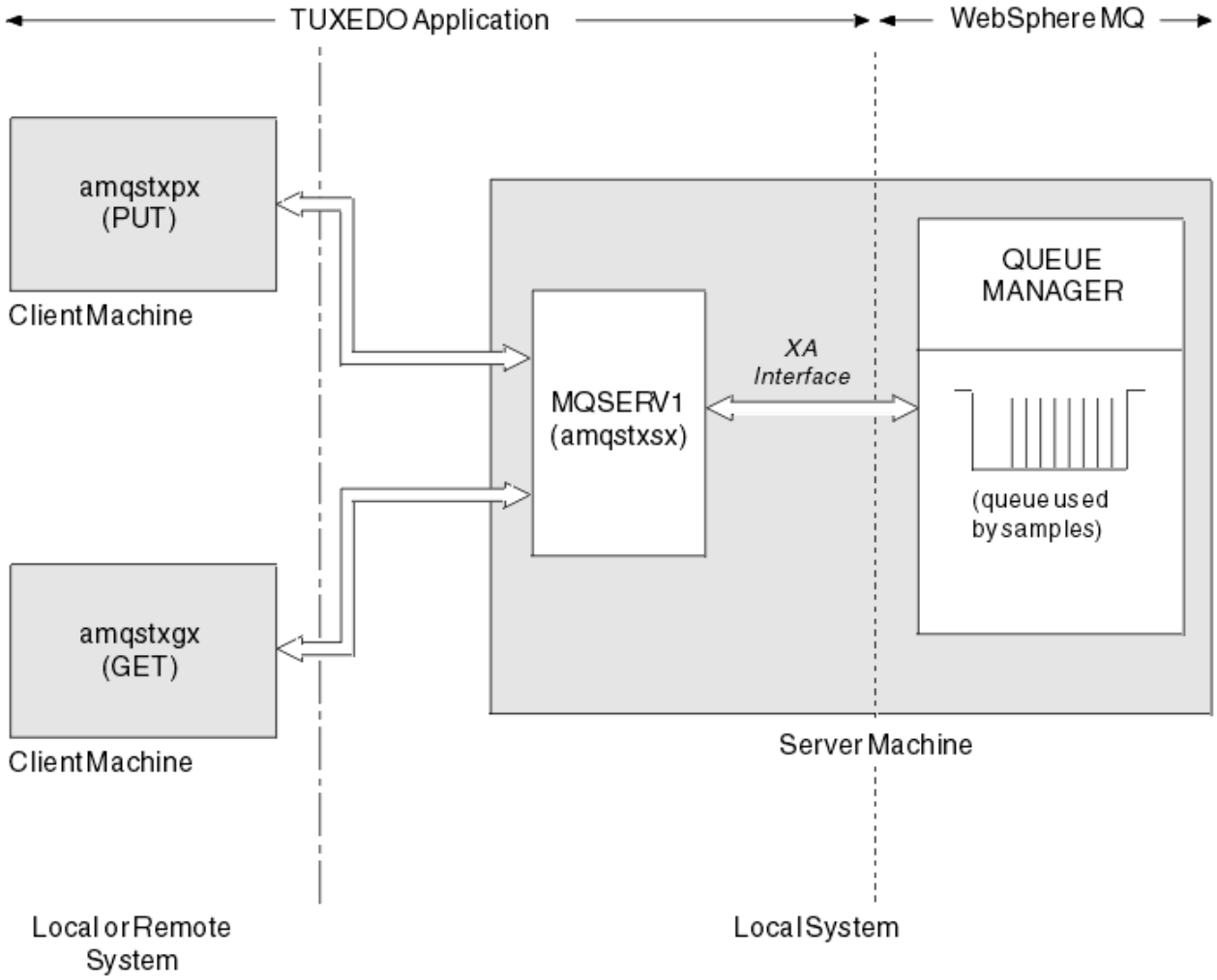


Figura 28. Cómo funcionan juntos los ejemplos de TUXEDO

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.

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 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:

- `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.

Utilización de la salida de seguridad SSPI en sistemas 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. Tanto para el cliente como para el servidor, se instala como una parte estándar de WebSphere MQ para Windows en la carpeta *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* . Por ejemplo, `C:\Program Files\IBM\WebSphere 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 DOMAIN\user. 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 suministrados entre sistemas que operan dentro de un dominio de seguridad Kerberos , cree un nombre servicePrincipalpara el gestor de colas.

Código fuente

El archivo de código fuente de salida se denomina amqsspin.c. Está en `C:\Program Files\IBM\WebSphere 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.exey la biblioteca Visual C++ y la carpeta `include` estén disponibles en la vía de acceso. Por ejemplo:

```
cl /VERBOSE /LD /MT /I<path_to_Microsoft_platform_SDK\include>
/I<path_to_WebSphere MQ\tools\c\include> amqsspin.c /DSECURITY_WIN32
-link  /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

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 WebSphere 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 WebSphere 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.

Para una cola en clúster con varias instancias en un clúster, WebSphere MQ utiliza la propiedad de prioridad de carga de trabajo de clúster **CLWLPRTY** de cada instancia de cola para determinar a qué instancias enviar mensajes. WebSphere 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 agrupación en clúster WebSphere 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 (utilizando **MQGET**) y colocando mensajes (utilizando **MQPUT**) en la misma cola de clúster. Esta transferencia hace que la gestión de carga de trabajo de clúster de WebSphere MQ seleccione una instancia de destino diferente, basándose en un valor CLWLPRTY superior al de la instancia de cola local. 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 WebSphere MQ subyacente y de 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 debe ejecutarse con la autorización adecuada para acceder a todos los recursos de WebSphere 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 WebSphere MQ enviará mensajes a las instancias de colas en clúster que residen en el mismo gestor de colas al que está conectada una aplicación de colocación. 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 de WebSphere MQ no deben utilizar **CLWLPRTY** para las colas que se van a supervisar o 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

Para ejecutar el ejemplo de supervisión de colas de clúster, debe configurar el gestor de colas para que acepte de forma segura las solicitudes de conexión entrantes de las aplicaciones que se ejecutan en modalidad de cliente.

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 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 WebSphere 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 siguiente ejemplo de **RUNMQSC** para definir el ejemplo de supervisión de colas de clúster como un servicio de IBM WebSphere MQ .

   ```
   define service(AMQSCLM) +
       descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
       control(qmgr) +
       servtype(server) +
       startcmd('<Install Root>\tools\c\samples\Bin\AMQSCLM.exe') +
    startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l c:\AMQSCLM\rpts') +
    stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
    stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

   donde < Instalar raíz > es la ubicación de la instalación.

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.

Definición	Descripción
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**	Vía de acceso completa y nombre de archivo de un archivo que contiene una lista de nombres de cola de máscaras de nombre de cola a 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 de directorio en la que se almacenan los mensajes de información registrados en un conjunto de < fn> envoltorios Para cada combinación de gestor de colas y cola se genera un archivo de informe que se limita a un determinado tamaño. El registrador siempre escribe en el mismo archivo, pero también mantiene las dos versiones anteriores del archivo. < /fn> archivos de informe.
-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) Cambia automáticamente la propiedad **CLWLUSEQ** de una instancia de cola supervisada a ANY cuando está inactiva y a la propiedad **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.

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 WebSphere 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 WebSphere 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 mensajes de una cola inactiva en una instancia activa, o instancias

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 supere 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**=QMGRy el argumento **-u** no está establecido.

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 las instancias de la cola con un valor de **CLWLPRTY**de 1 o superior.

Explicación potencial: Las instancias de cola remota tienen consumidores (**IPPROCS**> = 1) pero están inactivas en esos gestores de colas (**CLWLPRTY**= 0) porque AMQSCLM no está supervisando esas 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 potencial: Las instancias de cola remota están activas (**CLWLPRTY**= 1), pero se ven 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: Este escenario podría deberse a que AMQSCLM no se está ejecutando como 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.

Programa de ejemplo para Connection Endpoint Lookup (CEPL)

El ejemplo Búsqueda de puntos finales de conexión de IBM WebSphere MQ proporciona un módulo de salida sencillo pero potente que ofrece a los usuarios de WebSphere MQ una forma de recuperar definiciones de conexión de un repositorio LDAP como Tivoli Directory Server.

Tivoli Directory Server v6.3 Client debe estar instalado para poder utilizar CEPL.

Para utilizar este ejemplo es necesario tener conocimientos prácticos de la administración de WebSphere MQ en las plataformas soportadas.

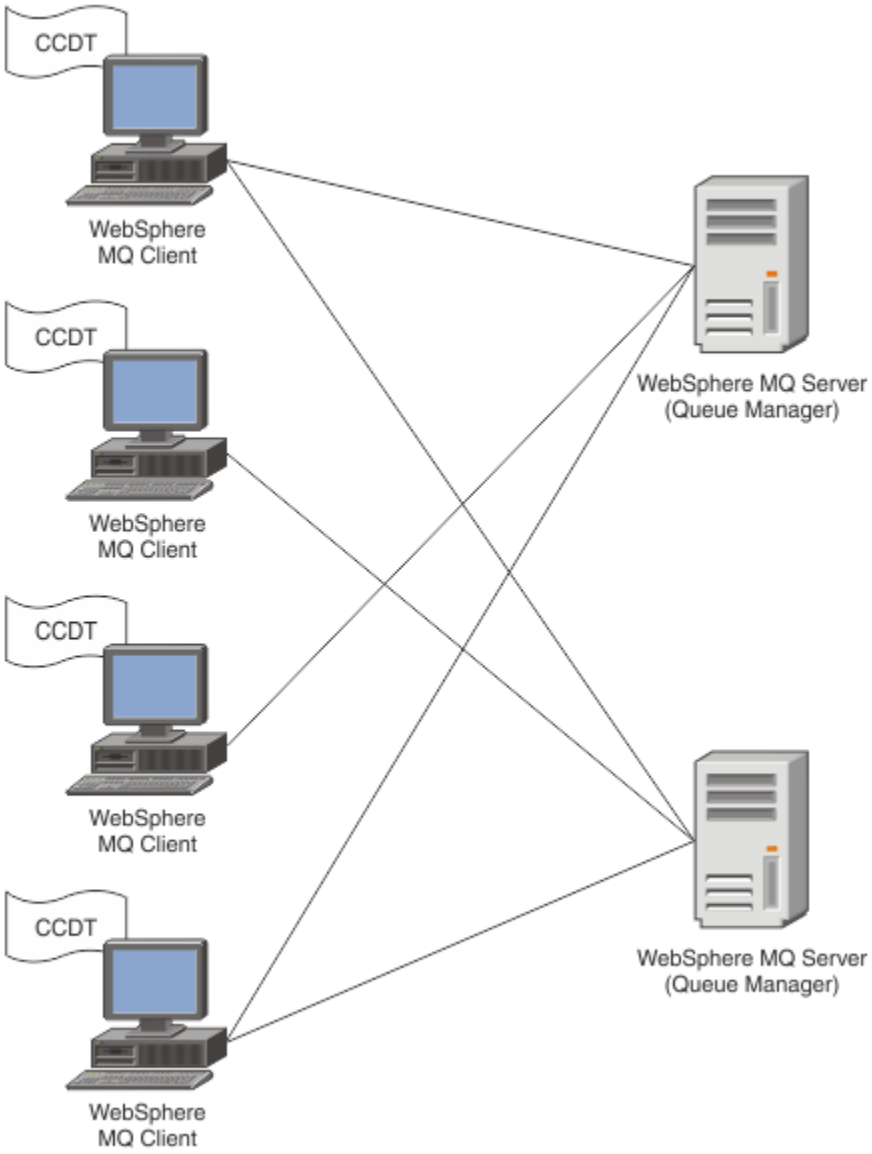
Introducción

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 WebSphere 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 de administración MQSC estándar de WebSphere 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.

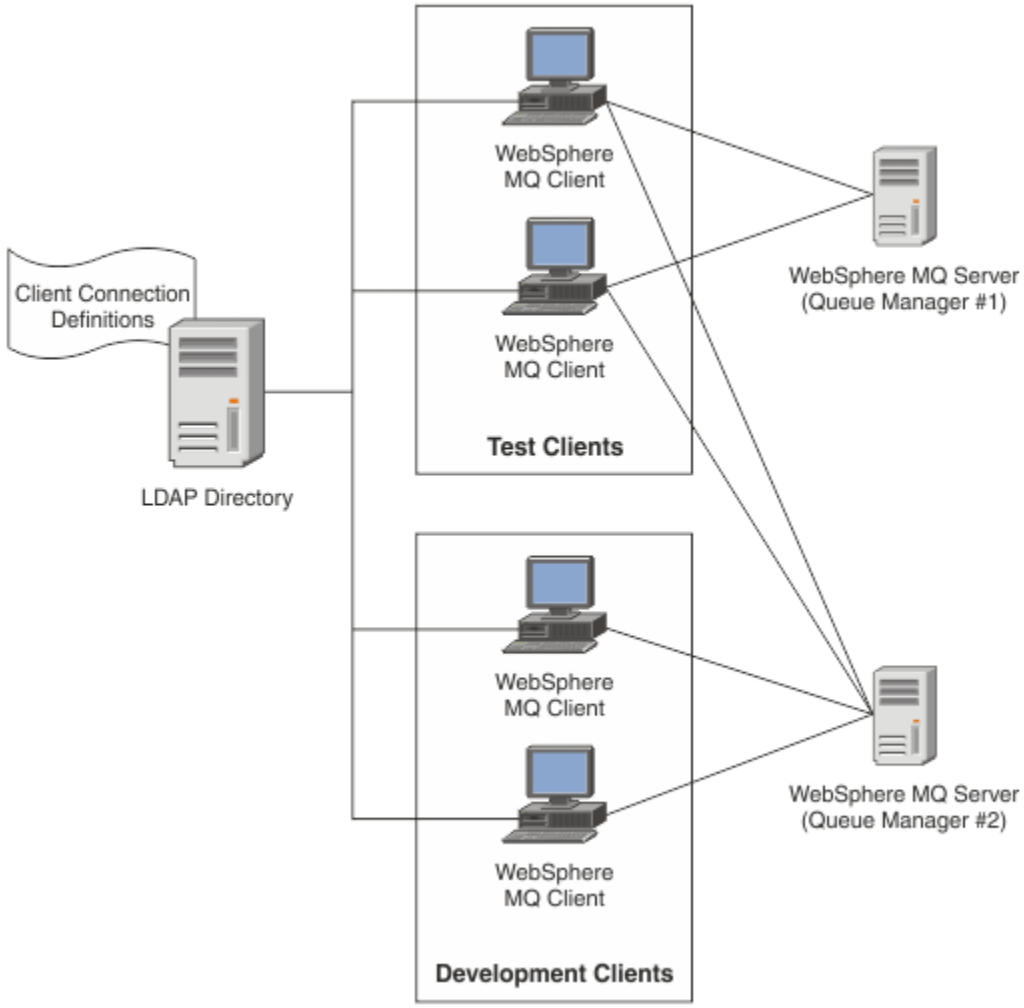


El archivo CCDT debe distribuirse a cada cliente WebSphere MQ . En los casos en que haya miles de clientes en local o de forma global, pronto será difícil de mantener y administrar. Se necesita un enfoque más flexible para garantizar que cada cliente tenga a su disposición las definiciones de cliente 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 a los gestores

de cola 1 y 2, mientras que los clientes de desarrollo pueden acceder únicamente al gestor de cola 2.



El módulo de salida puede buscar un repositorio LDAP, por ejemplo, IBM Tivoli Directory Server, para recuperar definiciones de canal. Utilizando estas definiciones de conexión, una aplicación cliente WebSphere MQ puede establecer 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 WebSphere 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.

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 WebSphere MQ requiere el software siguiente:

- IBM WebSphere MQ V7.0, o posterior
- Cliente de Tivoli Directory Server V6.3 o posterior

Sistemas operativos soportados:

1. Windows (XP/2003/2008)
2. Solaris (SPARC y x86-64)
3. AIX
4. Linux
   - RHEL v4 y v5 en System p
   - SUSE v9 y v10 en System p
   - RHEL v4 y v5 System x32 bit y x64 bit
   - SUSE v9 y v10 System x32 bit y x64 bit
5. HP IA64.

Nota: El programa de ejemplo no está disponible para las plataformas z/OS, i/5y HP PARISC.

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 WebSphere 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/<install name>/` antes de que se pueda utilizar. Para plataformas de 64 bits, el módulo de salida debe copiarse en exit64/<installation >/antes de que se pueda utilizar.

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 presupone que se está utilizando IBM Tivoli Directory Server (ITDS).

- Asegúrese de que IBM Tivoli Directory Server se esté ejecutando y, a continuación, 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 el ID de LDAP y la contraseña LDAP son el DN raíz y la contraseña del servidor LDAP:

  ldapadd -D "ID LDAP" -w "contraseña LDAP" -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, el archivo *mqclient.ini* . La sección PreConnect contiene las siguientes palabras clave:

Módulo : el nombre del módulo que contiene el código de salida de API. Si este campo contiene la vía de acceso completa del módulo, se utiliza tal cual *exit* o se busca en la carpeta *exit64* de la instalación de WebSphere MQ .

Función : Nombre del punto de entrada funcional en la biblioteca que contiene el código de salida PreConnect . La definición de función cumple el prototipo MQ_PRECONNECT_EXIT.

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=PreConnectExit
  Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
  Sequence=1
```

descripción general de salida y esquema

Sintaxis y parámetros utilizados para establecer una conexión con un gestor de colas.

WebSphere MQ v7.5 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, WebSphere MQ C Client 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-Finalización satisfactoria

MQCC_WARNING-Aviso (finalización parcial)

MQCC_FAILED-Error de llamada

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.

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 */
};
```

Código de ejemplo para crear una salida de búsqueda de punto final de conexión

Fragmentos de código para compilar el origen en Windows y plataformas distribuidas.

Compilación del código fuente

Puede compilar el origen con cualquier biblioteca de cliente LDAP, por ejemplo, IBM Tivoli Directory Server 6.3 Bibliotecas de cliente. En esta documentación se presupone que está utilizando bibliotecas de cliente de Tivoli Directory Server 6.3 .

Nota: La biblioteca de salida previa a la conexión se ha probado con los siguientes servidores LDAP:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Los siguientes fragmentos de código describen cómo compilar las salidas en Windowsy otras plataformas distribuidas:

Compilación de la salida en la plataforma Windows

Puede utilizar el siguiente fragmento de código para compilar el origen de salida en Windows:

```
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: Es posible que reciba avisos al compilar las bibliotecas de cliente de IBM Tivoli Directory Server 6.3 con el compilador Microsoft Visual Studio 2005 o superior, si utiliza las bibliotecas de cliente de IBM Tivoli Directory Server 6.3 compiladas con el compilador Microsoft Visual Studio 2003.

Compilación de la salida en otras plataformas distribuidas

Puede utilizar el siguiente fragmento de código para compilar el origen de salida en otras plataformas distribuidas, por ejemplo, Linux. Algunas opciones de compilador pueden diferir en otras plataformas distribuidas.

```
#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 proporciona bibliotecas de enlace estático y dinámico, pero sólo se puede utilizar un formato de las bibliotecas. 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)
```

Invocación de módulo de salida

El módulo de salida PreConnect se puede invocar con tres códigos de razón diferentes. Esta sección describe cada razón de salida en mayor profundidad.

MQXR_INIT

La salida se invoca con el código de razón MQXR_INIT para inicializar y establecer conexión con un servidor LDAP.

Antes de la llamada *MQXR_INIT* , el campo *pExitDataPtr* de la estructura MQNXP se habría rellenado con el atributo Data de la stanza PreConnect dentro del archivo *mqclient.ini* (es decir, LDAP).

Un URL de LDAP consta como mínimo del protocolo, nombre de host, número de puerto y DN base para la búsqueda. La salida analiza el URL de LDAP contenido en el campo *pExitDataPtr* , asigna una estructura de contexto de búsqueda LDAP MQNLDAPCTX y la rellena en consecuencia. La dirección de

esta estructura se almacena en el campo *pExitUserAreaPtr* . Si no se analiza correctamente el URL de LDAP, se produce el error MQCC_FAILED.

En este punto, la salida se conecta y 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 parámetro *QMgrName* contiene un nombre de gestor de colas específico, la búsqueda devuelve todas las definiciones de canal cuyo valor de atributo LDAP *ibm-amqQueueManagerName* coincide con el nombre de gestor de colas especificado.

Si el parámetro *QMgrName* es '*' o ' ' (en blanco), la búsqueda devuelve todas las definiciones de canal cuyo atributo de punto final de conexión *ibm-amqIsClientDefault* 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 operación, la salida se desconecta del servidor LDAP, libera toda la memoria asignada y mantenida por la salida. Esto incluirá la estructura *MQNLDAPCTX* , la matriz de punteros y cada MQCD al que hace referencia. Cualquier otro campo se establece en los valores predeterminados. Los parámetros de salida *pQMgrName* y *ppConnectOpts* no se utilizan durante *MQXR_TERM* y pueden ser NULL.

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 WebSphere MQ utiliza un directorio LDAP para obtener las definiciones de conexión. La estructura de las definiciones de conexión de cliente de WebSphere MQ dentro del 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 WebSphere MQ se describen en Atributos LDAP .

atributos LDAP

Los atributos LDAP definidos son específicos de WebSphere MQ y se correlacionan directamente con las propiedades de conexión de cliente.

WebSphere MQ Atributos de serie de directorio de canal de cliente

Los atributos de serie de caracteres con su correlación con las propiedades de WebSphere 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 21. Atributos de serie de directorio de canal de cliente de WebSphere MQ

Atributo LDAP	Descripción	Propiedad de WebSphere 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	es un nombre de modalidad bThe LU 6.2 .	MODENAME
ibm-amqPassword	La contraseña que se puede utilizar.	CONTRASEÑA
ibm-amqQueueManagerName	El nombre del gestor de colas o grupo de gestores de colas al que una aplicación cliente de WebSphere MQ puede solicitar 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 única CipherSpec para una conexión SSL.	SSLCIPH
ibm-amqSslPeerName	Comprueba el nombre distinguido (DN) del certificado del gestor de colas o cliente de igual en el otro extremo de un canal de WebSphere 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 entero de conexión de cliente de WebSphere 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 22. Atributos de entero de directorio de canal de cliente de WebSphere MQ		
Atributo LDAP	**Descripción**	**Propiedad de WebSphere 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 WebSphere MQ

Este atributo booleano no está correlacionado con ninguna propiedad WebSphere MQ . La sintaxis de este atributo indica un valor booleano.

Tabla 23. Atributo booleano de canal de cliente WebSphere 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 lista de canales de cliente de WebSphere MQ

Las propiedades de WebSphere MQ se almacenan como un atributo de lista separado por comas de valor único dentro del 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 WebSphere MQ se describen en la tabla siguiente.

Tabla 24. Atributos de lista de canales de cliente de WebSphere MQ		
Atributo LDAP	**Descripción**	**Propiedad de WebSphere 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

Tabla 24. Atributos de lista de canales de cliente de WebSphere MQ (continuación)		
Atributo LDAP	**Descripción**	**Propiedad de WebSphere MQ**
ibm-amqReceiveExitName	El nombre del programa de salida de usuario que va a ejecutar la salida de usuario de recepción de canal.	RCVEXIT

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.

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 subserie es una regla de coincidencia utilizada en el subesquema que especifica el comportamiento del atributo en un filtro de búsqueda, utilizando una subserie y contiene uno o varios comodines.

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.

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.

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.

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.

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.

ibm-amqQueueManagerName
Este atributo especifica el nombre del gestor de colas o del grupo de gestores de colas al que una aplicación cliente de WebSphere MQ puede solicitar 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.

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.

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.

ibm-amqSslCipherSpec
Este atributo LDAP especifica una sola CipherSpec para una conexión SSL.

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.

ibm-amqSslPeerName
Este atributo LDAP se utiliza para comprobar el nombre distinguido (DN) del certificado del gestor de colas de igual o del cliente en el otro extremo de un canal de WebSphere 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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, los valores especificados para **`ibm-amqHeaderCompression`** son `0` que se correlaciona con `NONE`. El cliente ignorará los valores que excedan el límite máximo permitido. Por ejemplo, ibm-amqHeaderCompression contiene un máximo de 2 enteros en la lista.

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 no tiene en cuenta los valores que excedan el límite máximo permitido. Por ejemplo, ibm-amqMessageCompression contiene un máximo de 16 enteros en la lista.

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.

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.

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.

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 es 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.

Escritura de una aplicación de gestión de colas

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:

Conceptos relacionados

"Conceptos de desarrollo de aplicaciones" en la página 8
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

"Decidir qué lenguaje de programación utilizar" en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

"Diseño de aplicaciones IBM WebSphere MQ" en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

"Programas WebSphere MQ de ejemplo" en la página 98
Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

"Escritura de aplicaciones cliente" en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

"Utilización de servicios web en WebSphere MQ" en la página 965
Puede desarrollar aplicaciones IBM WebSphere MQ para servicios web utilizando el transporte IBM WebSphere MQ para SOAP o el puente IBM WebSphere MQ para HTTP.

"Creación de una aplicación IBM WebSphere MQ" en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

"Manejo de errores de programa" en la página 559
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.

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.

WebSphere MQ para Windows y WebSphere MQ en sistemas UNIX and Linux también proporcionan:

- Llamadas a través de las cuales los programas WebSphere MQ para Windows y WebSphere MQ en sistemas UNIX and Linux 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 "Programas de ejemplo para plataformas distribuidas" en la página 98.
- Código de ejemplo fuente y ejecutable para enlazar con los gestores de transacciones externos.

Utilice los enlaces siguientes para obtener más información sobre MQI:

Conceptos relacionados

"Conexión y desconexión de un gestor de colas" en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

"Colocación de mensajes en una cola" en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

"Confirmación y restitución de unidades de trabajo" en la página 330
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 WebSphere MQ utilizando desencadenantes" en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

llamadas MQI

Utilice esta información para obtener más información sobre las llamadas en 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 WebSphere 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. 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 de IBM WebSphere MQ en plataformas Windows, UNIX y Linux

Los siguientes productos proporcionan las llamadas MQCMIT y MQBACK:

- IBM WebSphere MQ para Windows
- IBM WebSphere MQ en sistemas UNIX and Linux

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 cambios en el entorno CICS , utilice mandatos como EXEC CICS SYNCPOINT y EXEC CICS SYNCPOINT ROLLBACK.

Conversión de datos, tipos de datos, definiciones de datos y estructuras

Utilice esta información para obtener información sobre conversiones de datos, tipos de datos elementales, definiciones de datos de WebSphere MQ y estructuras cuando se utiliza la interfaz de cola de mensajes.

Conversión de datos

La llamada MQXCNVC (convertir caracteres) convierte los datos de carácter de un mensaje de un juego de caracteres a otro. Excepto en WebSphere MQ para z/OS, esta llamada sólo se utiliza desde 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 424 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 de WebSphere MQ

Los archivos de definición de datos proporcionados con WebSphere MQ contienen:

- Definiciones de todas las constantes y códigos de retorno de WebSphere MQ
- Definiciones de estructuras y tipos de datos de WebSphere 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 definición de datos de WebSphere MQ , consulte “IBM WebSphere MQ archivos de definición de datos” en la página 82.

Estructuras

Las estructuras que se usan en las llamadas MQI listadas en “llamadas MQI” en la página 200 se proporcionan en los archivos de definición de datos de cada lenguaje de programación soportado.

Consulte Resumen de tipos de datos de estructuras para obtener un resumen de las estructuras.

Archivos de biblioteca y programas de apéndice de IBM WebSphere 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 IBM WebSphere MQ” en la página 438. Para obtener información sobre cómo enlazar con archivos de biblioteca C++, consulte Utilización de C++ *WebSphere MQ Utilización de C++*.

IBM WebSphere MQ para Windows

En IBM WebSphere MQ para Windows, debe enlazar el programa con los archivos de biblioteca MQI proporcionados para el entorno en el que está ejecutando la aplicación, además de los proporcionados por el sistema operativo:

Tabla 25. 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)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib	Soporte de servidor TXSeries CICS para C (32 bits)

Tabla 25. Archivos de biblioteca para aplicaciones Windows (continuación)	
Archivo de biblioteca	**Entorno**
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib	Soporte de cliente TXSeries CICS 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 WebSphere MQ.

Utilice `amqmdnet.dll` para compilar programas .NET. Consulte “Compilación de programas WebSphere MQ .NET” en la página 606 en la sección “Utilización de .NET” en la página 571 para obtener más información.

Estos archivos se suministran por motivos de compatibilidad con releases anteriores:

mqic32.lib
mqic32xa.lib

IBM WebSphere MQ para AIX

En IBM WebSphere MQ para AIX, debe enlazar el programa con los archivos de biblioteca MQI proporcionados para el entorno en el que está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

En una aplicación sin hebras:

Tabla 26. Archivos de biblioteca para aplicaciones AIX sin hebras

Archivo de biblioteca	**Entorno**
libmqm.a	Servidor en C
libmqic.a & **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	Biblioteca de tiempo de ejecución de WebSphere MQ para soporte de Micro Focus COBOL
libmqmcb.a	Servidor para COBOL
libmqicb.a	Cliente para COBOL
libimqc23ia.a	Cliente en C++
libimqs23ia.a	Servidor en C++

En una aplicación con hebras:

Tabla 27. Archivos de biblioteca para aplicaciones AIX con hebras

Archivo de biblioteca	**Entorno**
libmqm_r.a	Servidor en C
libmqic_r.a & **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 en C++
libimqs23ia_r.a	Servidor en C++

IBM WebSphere MQ para HP-UX

En IBM WebSphere MQ para HP-UX, debe enlazar el programa con los archivos de biblioteca MQI proporcionados para el entorno en el que está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

Plataforma IA64 (IPF)

En una aplicación sin hebras:

Tabla 28. Archivos de biblioteca para aplicaciones HP-UX sin hebras	
Archivo de biblioteca	**Entorno**
libmqm.so	Servidor en C
libmqic.so & **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
libimqi23ah.so	C++
libmqmcbrt.o	Biblioteca de tiempo de ejecución de WebSphere MQ para soporte de Micro Focus COBOL
libmqmcb.so	Servidor para COBOL
libmqicb.so	Cliente para COBOL

En una aplicación con hebras:

Tabla 29. Archivos de biblioteca para aplicaciones HP-UX con hebras	
Archivo de biblioteca	**Entorno**
libmqm_r.so	Servidor en C
libmqmzf_r.so & **libmqm_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
libimqi23ah_r.so	C++

IBM WebSphere MQ para Linux

En IBM WebSphere 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:

Tabla 30. Archivos de biblioteca para aplicaciones Linux sin hilos	
Archivo de biblioteca	**Entorno**
libmqm.so	Servidor en C
libmqic.so & **libmqm.so**	Cliente en C
libmqmzf.so	Salidas de servicio instalable en C
libmqmxa.so	Interfaz XA de servidor

Tabla 30. Archivos de biblioteca para aplicaciones Linux sin hilos (continuación)	
Archivo de biblioteca	**Entorno**
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:

Tabla 31. Archivos de biblioteca para aplicaciones Linux con hebras	
Archivo de biblioteca	**Entorno**
libmqm_r.so	Servidor en C
libmqic_r.so & **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++

IBM WebSphere MQ Para Solaris

En IBM WebSphere MQ para Solaris, debe enlazar el programa con los archivos de biblioteca MQI proporcionados para el entorno en el que está ejecutando la aplicación, además de los proporcionados por el sistema operativo.

Tabla 32. Archivos de biblioteca para aplicaciones Solaris	
Archivo de biblioteca	**Entorno**
libmqm.so	Servidor y cliente en C
libmqmzse.so	En C
libmqic.so	Cliente en C
libmqmcs.so	Servicios comunes 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
libimqc23as.a	Cliente en C++
libimqs23as.a	Servidor en C++

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 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 WebSphere MQ , el programa debe tener un identificador exclusivo por el que conozca 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:

- Códigos de razón para todas las demás plataformas WebSphere 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.

Consideraciones sobre UNIX and Linux

Consideraciones que hay que tener en cuenta.

Tenga en cuenta que los puntos siguientes al desarrollar aplicaciones UNIX and Linux.

La llamada de sistema fork en sistemas UNIX and Linux

Tenga en cuenta estas consideraciones cuando utilice una llamada al sistema fork en aplicaciones IBM WebSphere MQ.

Si una aplicación necesita usar `fork`, su proceso padre tendrá que invocar `fork` antes de que se efectúe cualquier llamada IBM WebSphere MQ, por ejemplo, `MQCONN` o se cree un objeto IBM WebSphere MQ usando **`ImqQueueManager`**.

Si la aplicación necesita crear un proceso hijo tras efectuar una llamada IBM WebSphere 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 WebSphere MQ realizada en el proceso hijo devuelve MQRC_ENVIRONMENT_ERROR.

Manejo de señales de UNIX and Linux

Esto no se aplica a WebSphere MQ para z/OS o WebSphere MQ para Windows.

En general, los sistemas UNIX, Linux y IBM i han pasado 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.

Aplicaciones sin hebras

No es aplicable en Solaris, ya que todas las aplicaciones se consideran con hebras incluso si utilizan una sola hebra.

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.

Después de que WebSphere MQ maneje 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.

Aplicaciones con hebras

Se considera que una hebra está conectada a WebSphere MQ desde MQCONN (o MQCONNX) hasta MQDISC.

Señales síncronas

Una señal síncrona surge en una hebra concreta.

Los sistemas UNIX and Linux permiten de forma segura configurar un manejador de este tipo de señales para para todo el proceso. Sin embargo, WebSphere MQ configura su propio manejador para las señales siguientes, en el proceso de aplicación, mientras cualquier hebra está conectada a WebSphere MQ:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Si está escribiendo aplicaciones multihebra, solo hay un manejador de señales a nivel de proceso por cada señal. Cuando WebSphere MQ configura sus propios manejadores de señales síncronas, guarda los manejadores registrados anteriormente para cada señal. Después de que WebSphere MQ maneje una de las señales listadas, WebSphere MQ intenta llamar al manejador de señales que estaba en vigor en el momento de la primera conexión de WebSphere MQ dentro del proceso. Los manejadores registrados anteriormente se restauran cuando todas las hebras de aplicación se han desconectado de WebSphere MQ.

Puesto que los manejadores de señales se guardan y restauran mediante WebSphere 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 WebSphere MQ.

Nota: Cuando una aplicación, o una biblioteca de middleware (que se ejecuta como parte de una aplicación), establece un manejador de señales mientras una hebra está conectada a WebSphere MQ, el manejador de señales de la aplicación debe llamar al manejador WebSphere MQ correspondiente durante el proceso 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 WebSphere MQ, el manejador de señales anterior debe restaurarse antes de que la aplicación se desconecte de WebSphere MQ.
- Cuando una aplicación establece un manejador de señales antes de conectarse a WebSphere MQ, la aplicación debe desconectarse de WebSphere 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

WebSphere MQ no utiliza ninguna señal asíncrona en aplicaciones con hebras a menos que sean aplicaciones cliente.

Consideraciones adicionales sobre las aplicaciones cliente con hebras

WebSphere MQ maneja las señales siguientes durante la E/S en 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
Tenga en cuenta estas consideraciones al utilizar el manejo de señales de UNIX .

Aplicaciones Fastpath (de confianza)

Las aplicaciones de vía de acceso rápida se ejecutan en el mismo proceso que WebSphere MQ y, por lo tanto, se ejecutan en el entorno multihebra.

En este entorno, WebSphere MQ maneja las señales síncronas SIGSEGV, SIGBUS, SIGFPE y SIGILL. Todas las demás señales no se deben entregar a la aplicación Fastpath mientras esté conectada a WebSphere 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 213.

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 (), WebSphere MQ restituye los mensajes no confirmados en el punto de sincronización como de costumbre 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.

En el entorno AIX , WebSphere MQ requiere que se reinicien las llamadas al sistema interrumpidas por 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 acciones; de lo contrario, es posible que WebSphere MQ no pueda 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 de WebSphere MQ en un entorno multihebra tienen las mismas restricciones que para las aplicaciones de vía de acceso rápida. Considere que están permanentemente conectados a WebSphere MQ y, por lo tanto, no utilizan señales o llamadas de sistema operativo que no sean seguras en ejecución multihebra.

Manejadores de salidas MVS

Los usuarios pueden instalar manejadores de salida para una aplicación WebSphere MQ utilizando el servicio del sistema **SYS$DCLEXH** .

El manejador de salidas recibe el control cuando sale una imagen. Una salida de imagen suele tener lugar cuando se invocan los servicios Exit ($EXIT) o Force Exit ($FORCEX). $FORCEX interrumpe el proceso de destino en modo de usuario. A continuación, todos los manejadores de salida en modo usuario (establecidos por $DCLEXH) empiezan a ejecutarse en orden inverso al establecimiento. Para

obtener más detalles sobre los manejadores de salidas y $FORCEX, consulte el *Manual de conceptos de programación VMS* y el *Manual de servicios del sistema de VMS*.

Si llama a una función MQI desde dentro de un manejador de salidas, el comportamiento de la función dependerá de la forma en que se haya terminado la imagen. Si la imagen se ha terminado mientras otra función MQI está activa, se devolverá un `MQRC_CALL_IN_PROGRESS`.

Es posible llamar a una función MQI desde dentro de un manejador de salida si ninguna otra función MQI está activa y las llamadas de actualización están inhabilitadas para la aplicación WebSphere MQ . Si las llamadas de actualización están habilitadas para la aplicación WebSphere MQ , falla con el código de razón `MQRC_HCONN_ERROR`.

El ámbito de una llamada MQCONN o MQCONNX suele ser la hebra que la ha emitido. Si las llamadas ascendentes están habilitadas, el manejador de salidas ejecutará una hebra aparte y los descriptores de conexión no se podrán compartir.

Los manejadores de salidas se inician dentro del contexto interrumpido del proceso de destino. Es la aplicación la que tiene que asegurarse de que las acciones realizadas por un manejador sean seguras y fiables para el contexto interrumpido asíncronamente desde el que se invocan.

Conexión y desconexión de un gestor de colas

Para utilizar los servicios de programación de WebSphere 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:

z/OS por lotes, WebSphere MQ para IBM i, WebSphere MQ en sistemas UNIX , WebSphere MQ en sistemas Linux y WebSphere MQ para Windows

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.

Los programas por lotes z/OS pueden conectarse, de forma consecutiva o simultánea, a varios gestores de colas en el mismo TCB.

IMS

La región de control IMS se conecta a uno o varios gestores de colas cuando se inicia. Esta conexión está controlada por mandatos IMS . Sin embargo, los grabadores de programas IMS de colas de mensajes deben utilizar la llamada MQCONN MQI para especificar el gestor de colas al que desean conectarse. 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.

Los programas IMS pueden conectarse, de forma consecutiva o simultánea, a varios gestores de colas en el mismo TCB.

CICS Transaction Server para z/OS y CICS para MVS/ESA

Los programas CICS no necesitan realizar ningún trabajo para conectarse a un gestor de colas porque el propio sistema CICS está conectado. Normalmente, esta conexión se realiza automáticamente durante la inicialización, pero también puede utilizar la transacción CKQC, que essuministrado con WebSphere MQ para z/OS.

Las tareas de CICS solo se pueden conectar al gestor de colas al que está conectada la propia región CICS .

Nota: Los programas CICS también pueden utilizar las llamadas de conexión y desconexión MQI (MQCONN y MQDISC). Es posible que desee hacerlo para que pueda portar estas aplicaciones a entornos no CICS con un mínimo de recodificación. Sin embargo, 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 sistemas abiertos
Estos programas no necesitan realizar ningún trabajo para conectarse a un gestor de colas porque 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 212
- "Conexión a un gestor de colas mediante la llamada MQCONNX" en la página 213
- "Desconectar programas desde un gestor de colas utilizando MQDISC" en la página 218

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Apertura y cierre de objetos" en la página 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

"Colocación de mensajes en una cola" en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

"Confirmación y restitución de unidades de trabajo" en la página 330
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 WebSphere MQ utilizando desencadenantes" en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

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 WebSphere MQ para z/OS, en el entorno de proceso por lotes, el gestor de colas predeterminado se especifica en el módulo CSQBDEFV.
- Para los sistemas IBM WebSphere MQ para Windows, IBM i, UNIXy Linux , el gestor de colas predeterminado se especifica en el archivo mqs.ini .

De forma alternativa, en los entornos por lotes, TSO y RRS de z/OS MVS , 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. Tiene que pertenecer al mismo sistema que la aplicación IBM WebSphere 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 WebSphere MQ para z/OS .

Con el entorno TXSeries CICS y TXSeries para Windows y AIX, el gestor de colas debe definirse como un recurso XA en 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. Para obtener más información, consulte Seguridad.

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 220).

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 WebSphere 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, sólo se puede realizar una conexión de WebSphere MQ desde una hebra a la vez. De forma alternativa, considere “Conexiones (independientes de hebra) compartidas con MQCONNX” en la página 217 para permitir que se utilicen varias conexiones de WebSphere MQ desde una sola hebra y una conexión de WebSphere MQ desde cualquier hebra.[1]

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.

Como entrada de MQCONNX, puede proporcionar un nombre de gestor de colas o un nombre de grupo de compartición de colas en sistemas de colas compartidas z/OS . La salida de MQCONNX es:

- Un manejador de conexiones (Hconn)
- Un código de terminación

[1] Al utilizar aplicaciones multihebra con IBM WebSphere MQ en sistemas UNIX and Linux , debe asegurarse de que las aplicaciones tienen un tamaño de pila suficiente para las hebras. Considere la posibilidad de utilizar un tamaño de pila de 256 KB o superior cuando las aplicaciones de múltiples hebras estén realizando llamadas MQI, ya sea por sí mismas o con otros manejadores de señales (por ejemplo, CICS).

- Un código de razón

Puede utilizar el manejador de conexión en las llamadas MQI posteriores.

En MQCONNX se proporciona una descripción de todos los parámetros de MQCONNX. El campo *Options* le permite establecer STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING o ISOLATED_BINDING para cualquier versión de MQCNO. También puede crear conexiones (independientes de hebra) compartidas mediante una llamada MQCONNX. Consulte "Conexiones (independientes de hebra) compartidas con MQCONNX" en la página 217 para obtener más información al respecto.

MQCNO_STANDARD_BINDING

De forma predeterminada, MQCONNX (como MQCONN) implica dos hebras lógicas donde la aplicación WebSphere MQ y el agente del gestor de colas local se ejecutan en procesos separados. La aplicación WebSphere MQ solicita la operación WebSphere MQ y el agente del gestor de colas local da servicio a la solicitud. Esto se define mediante la opción MQCNO_STANDARD_BINDING en la llamada MQCONNX.

Si especifica MQCNO_STANDARD_BINDING, la llamada MQCONNX utiliza MQCNO_SHARED_BINDING o MQCNO_ISOLATED_BINDING, en función del valor del atributo de tipo DefaultBinddel gestor de colas, que se define en qm.ini o en el registro de Windows .

Éste es el valor predeterminado.

Si se va a enlazar a la biblioteca mqm, primero se intentará establecer una conexión de servidor estándar mediante el tipo de enlace predeterminado. Si no se ha podido cargar la biblioteca de servidor subyacente, en su lugar se intenta una conexión de cliente.

- Si se especifica la variable de entorno MQ_CONNECT_TYPE, se puede proporcionar una de las opciones siguientes para cambiar el comportamiento de MQCONN o MQCONNX si se especifica MQCNO_STANDARD_BINDING. La excepción a esto es si se especifica MQCNO_FASTPATH_BINDING con MQ_CONNECT_TYPE establecido en LOCAL o STANDARD, para permitir que el administrador degrade las conexiones de vía de acceso rápida sin que exista un cambio relativo en la aplicación:

Valor	Significado
CLIENT	Sólo se intenta una conexión de cliente.
FASTPATH	Este valor estaba soportado en los releases anteriores, pero ahora se pasa por alto si se ha especificado.
LOCAL	Sólo se intenta una conexión con el servidor. Las conexiones fastpath se reducen a una conexión con el servidor estándar.
ESTÁNDAR	Soportado por motivos de compatibilidad con releases anteriores. Este valor se trata ahora como LOCAL.

- Si la variable de entorno MQ_CONNECT_TYPE no está establecida cuando se invoca MQCONN, se intenta establecer una conexión de servidor estándar utilizando el tipo de enlace predeterminado. Si no se puede cargar la biblioteca del servidor, se intenta una conexión de cliente.

MQCNO_FASTPATH_BINDING

Las *aplicaciones de confianza* implican que la aplicación WebSphere 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. Esto se define mediante la opción MQCNO_FASTPATH_BINDING en la llamada MQCONNX.

Es necesario enlazar aplicaciones de confianza a las bibliotecas de WebSphere MQ con hebras. Para obtener instrucciones sobre cómo configurar una aplicación WebSphere MQ para que se ejecute como de confianza, consulte Opciones de MQCNO.

Esta opción proporciona el rendimiento más alto.

Nota: Esta opción pone en peligro la integridad del gestor de colas: no hay ninguna protección contra la sobrescritura de su almacenamiento. Esto también resulta aplicable si la aplicación contiene errores que se pueden exponer a los mensajes y también a los demás datos en el gestor de colas. Tenga en cuenta estos problemas antes de utilizar esta opción.

MQCNO_SHARED_BINDING

Especifique esta opción para que la aplicación y el agente del gestor de colas local se ejecuten en procesos separados. Esto mantiene la integridad del gestor de colas, es decir, se protege al gestor de colas contra los programas errantes. Sin embargo, la aplicación y el agente de gestor de colas local comparten algunos recursos.

Esta opción es intermedia entre MQCNO_FASTPATH_BINDING y MQCNO_ISOLATED_BINDING, tanto en términos de protección de la integridad del gestor de colas, como en términos del rendimiento de las llamadas MQI.

MQCNO_SHARED_BINDING se omite si el gestor de colas no soporta este tipo de enlace. El proceso continuará como si no se hubiese especificado la opción.

Si una aplicación se ha conectado al gestor de colas local utilizando MQCNO_SHARED_BINDING, se puede detener el gestor de colas mientras se ejecuta la aplicación. Si reinicia el gestor de colas mientras la aplicación está en ejecución, el inicio del gestor de colas falla con el error AMQ7018 ya que la aplicación todavía tiene retenidos recursos que necesita el gestor de colas.

Para iniciar el gestor de colas, debe detener la aplicación.

MQCNO_ISOLATED_BINDING

Especifique esta opción para que la aplicación y el agente del gestor de colas local se ejecuten en procesos separados, tal como ocurre con MQCNO_SHARED_BINDING. En este caso, sin embargo, el proceso de aplicación y el agente del gestor de colas local se aíslan entre sí en el sentido de que no comparten recursos.

Esta es la opción más segura para proteger la integridad del gestor de colas, pero proporciona el rendimiento más lento de las llamadas MQI.

Si el gestor de colas no da soporte a este tipo de enlace, se ignora MQCNO_ISOLATED_BINDING. El proceso continuará como si no se hubiese especificado la opción.

MQCNO_CLIENT_BINDING

Especifique esta opción para que la aplicación intente únicamente una conexión con el cliente. Esta opción tiene las siguientes limitaciones:

- MQCNO_CLIENT_BINDING se rechaza en z/OS con MQRC_OPTIONS_ERROR.
- MQCNO_CLIENT_BINDING se rechaza con MQRC_OPTIONS_ERROR, si se ha especificado con cualquier opción de enlace MQCNO distinta de MQCNO_STANDARD_BINDING.
- MQCNO_CLIENT_BINDING no está disponible para Java, ya que tiene sus propios mecanismos para seleccionar el tipo de enlace.
- V 7.5.0.7 Antes de IBM WebSphere MQ Version 7.5.0, Fixpack 7, MQCNO_CLIENT_BINDING no está disponible para .NET ya que tiene sus propios mecanismos para elegir el tipo de enlace. A partir de Version 7.5.0, Fix Pack 7, se elimina la restricción sobre la utilización de .NET para MQCNO_CLIENT_BINDING.
- Si la variable de entorno MQ_CONNECT_TYPE no se establece cuando se llama a MQCONNX, se intenta una conexión de servidor estándar utilizando el tipo de enlace predeterminado. Si no se puede cargar la biblioteca del servidor, se intenta una conexión de cliente.

MQCNO_LOCAL_BINDING

Especifique esta opción para hacer que la aplicación intente una conexión con el servidor. Si también se ha especificado MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING o MQCNO_SHARED_BINDING, la conexión será de dicho tipo, y se ha documentado en esta sección. De lo contrario, se intenta efectuar una conexión de servidor estándar utilizando el tipo de enlace predeterminado. MQCNO_LOCAL_BINDING tiene las limitaciones siguientes:

- MQCNO_LOCAL_BINDING se ignora en z/OS.
- MQCNO_LOCAL_BINDING se rechaza con MQRC_OPTIONS_ERROR si se ha especificado con cualquier opción de reconexión MQCNO que no sea MQCNO_RECONNECT_AS_DEF.
- MQCNO_LOCAL_BINDING no está disponible para Java, ya que tiene sus propios mecanismos para seleccionar el tipo de enlace.
- V 7.5.0.7 Antes de IBM WebSphere MQ Version 7.5.0, Fixpack 7, MQCNO_LOCAL_BINDING no está disponible para .NET ya que tiene sus propios mecanismos para elegir el tipo de enlace. A partir de Version 7.5.0, Fix Pack 7, se elimina la restricción sobre la utilización de .NET para MQCNO_LOCAL_BINDING.
- Si la variable de entorno MQ_CONNECT_TYPE no se establece cuando se llama a MQCONNX, se intenta una conexión de servidor estándar utilizando el tipo de enlace predeterminado. Si no se puede cargar la biblioteca del servidor, se intenta una conexión de cliente.

En z/OS estas opciones se toleran, pero sólo se realiza una conexión de enlace estándar. MQCNO Versión 3, para z/OS, permite cuatro opciones alternativas:

MQCNO_SERIALIZE_CONN_TAG_QSG

Esto permite que una aplicación solicite que se ejecute sólo una instancia de una aplicación, en cualquier momento, en un grupo de compartición de colas. Esto se consigue registrando el uso de un código de conexión con un valor que especifica o deriva la aplicación. El código es una serie de caracteres de 128 bytes, especificada en MQCNO Versión 3.

MQCNO_RESTRICT_CONN_TAG_QSG

Se utiliza cuando una aplicación consta de más de un proceso (o un TCB), cada uno de los cuales puede conectarse a un gestor de colas. La conexión solo se permite si no hay ningún uso actual del código, o si la aplicación solicitante se encuentra en el mismo ámbito de proceso. Se trata del espacio de direcciones MVS dentro del mismo grupo de compartición de colas que el propietario del código.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Esto es similar a MQCNO_SERIALIZE_CONN_TAG_QSG, pero solo se pregunta al gestor de colas local si ya se está utilizando el código solicitado.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Esto es similar a MQCNO_RESTRICT_CONN_TAG_QSG, pero solo se pregunta al gestor de colas local si ya se está utilizando el código solicitado.

Restricciones para aplicaciones de confianza

Se aplican las restricciones siguientes a las aplicaciones de confianza:

- Debe desconectar explícitamente las aplicaciones de confianza del gestor de colas.
- Debe detener las aplicaciones de confianza antes de finalizar el gestor de colas mediante 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.

- En sistemas WebSphere MQ en UNIX and Linux debe utilizar mqm como userID y groupID efectivos para todas las llamadas MQI. Puede cambiar estos ID antes de realizar una llamada no MQI que requiera autenticación (por ejemplo, abrir un archivo), pero *debe* cambiarlo de nuevo a mqm antes de realizar la siguiente llamada MQI.
- En WebSphere MQ para HP-UX, es probable que las aplicaciones de vía rápida multihebra necesiten establecer un tamaño de pila mayor que el predeterminado. Utilice un tamaño de 256 KB.
- En WebSphere MQ para aplicaciones de 64 bits de confianza de Windows no están soportadas. Si trata de ejecutar una aplicación fiable de 64 bits, ésta se degradará a una conexión de vínculo estándar.
- En WebSphere MQ en sistemas UNIX and Linux no se da soporte a 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 soportado en WebSphere MQ para z/OS.

En plataformas WebSphere MQ distintas de WebSphere MQ for z/OS, una conexión realizada con MQCONN sólo está disponible para 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 217 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.

Uso de opciones de llamada MQCONNX con MQ_CONNECT_TYPE

Utilice esta información para comprender las distintas opciones de llamada MQCONNX sobre cómo se utilizan con MQ_CONNECT_TYPE.

En WebSphere MQ para IBM i, WebSphere MQ para Windowsy WebSphere MQ en sistemas UNIX and Linux , 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 33. 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.

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 aplicaciones CICS Transaction Server para 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 WebSphere MQ para IBM i donde, al finalizar la sesión del sistema operativo, se realiza una llamada MQDISC implícita.

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 en z/OS, después de que se llame a MQDISC, el descriptor de conexión (Hconn) ya no es válido y no puede emitir más llamadas MQI hasta que vuelva a llamar a MQCONN o MQCONNX. MQDISC no ejecuta MQCLOSE de forma implícita para cualquier objeto que continúe abierto utilizando este descriptor.

Si utiliza MQCONNX para conectarse en WebSphere MQ para z/OS, MQDISC también finaliza el ámbito del código de conexión establecido por MQCONNX. Sin embargo, en una aplicación CICS, IMSo RRS, si hay una unidad de recuperación activa asociada a un código de conexión, MQDISC se rechaza con un código de razón de MQRC_CONN_TAG_NOT_RELEASE.

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 curso normal de los sucesos, un trabajo que tiene autorización para abrir o conectarse a un objeto WebSphere MQ se cierra o desconecta de dicho objeto. Incluso si se revoca la autorización de un trabajo que ha conectado o abierto un objeto de WebSphere MQ , se aceptan las llamadas MQCLOSE y MQDISC.

Apertura y cierre de objetos

Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

Para realizar cualquiera de las operaciones siguientes, primero debe *abrir* el objeto WebSphere MQ relevante:

- 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 238).

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 211.

Existen cuatro tipos de objeto WebSphere 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 WebSphere MQ , consulte Objetos.

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 240 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 empieza a procesar un nuevo usuario después de una llamada GU (get unique) 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 220
- “Creación de colas dinámicas” en la página 228
- “Apertura de colas remotas” en la página 229
- “Cierre de objetos utilizando la llamada MQCLOSE” en la página 229

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 211
Para utilizar los servicios de programación de WebSphere 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 230
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

“Confirmación y restitución de unidades de trabajo” en la página 330
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 WebSphere MQ utilizando desencadenantes” en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

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. Para aplicaciones CICS en z/OS, puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero) o utilizar el descriptor de conexión devuelto por 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 213 y la sección "Conexiones (independientes de hebra) compartidas con MQCONNX" en la página 217. No obstante, hay consideraciones adicionales para algunos entornos:

CICS
En un programa CICS , puede utilizar el descriptor de contexto sólo dentro de la misma tarea CICS desde la que ha realizado la llamada MQOPEN.

IMS y z/OS por lotes
En los entornos de IMS y por lotes, puede utilizar el descriptor de contexto dentro de la misma tarea, pero no dentro de 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 241 bajo "Listas de distribución" en la página 240.

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.

Al abrir una cola WebSphere 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 34 en la página 222 para obtener una visión general del proceso de resolución de nombres. El nombre que proporcione en *ObjectQMgrName* se resuelve *antes* que en *ObjectName*.

En la Tabla 34 en la página 222 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 tenga que especificar *ObjectName*.

Tabla 34. Resolución de nombres de cola cuando se utiliza MQOPEN

Entrada a MQOD		**Nombres resueltos**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Cola de transmisión**
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)
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.		

Tabla 34. Resolución de nombres de cola cuando se utiliza MQOPEN (continuación)

Entrada a MQOD		**Nombres resueltos**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Cola de transmisión**
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)
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)

Tabla 34. Resolución de nombres de cola cuando se utiliza MQOPEN (continuación)				
Entrada a MQOD		**Nombres resueltos**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**Cola de transmisión**
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. Cuando se utilizan gestores de colas de WebSphere MQ para z/OS que forman parte de un grupo de compartición de colas (QSG), se puede utilizar el nombre del QSG en lugar del nombre del gestor de colas local en Tabla 34 en la página 222.

   Si el gestor de colas local no puede abrir la cola de destino, o colocar un mensaje en la cola, el mensaje se transfiere al nombre ObjectQMgrespecificado a través de, ya sea una transferencia a colas dentro del grupo, o un canal WebSphere MQ .
6. En la columna *ObjectName* de la tabla, CLUSTER hace referencia a los atributos CLUSTER y CLUSNL de la cola.
7. La cola SYSTEM.QSG.TRANSMIT.QUEUE se utiliza si los gestores de colas locales y remotos están en el mismo grupo de compartición de colas; la transferencia a 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 sobre varias colas de transmisión de clúster, consulte Agrupación en clúster: Planificación de cómo configurar 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 de gestor de colas resuelto se almacenan en los campos *ResolvedQName* y *ResolvedQMgrName* en 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 más opciones para controlar el acceso que se le otorga al objeto que está abriendo. 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 descriptor de contexto de cola se toma del atributo de cola *DefBind* , que puede tomar el valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDo 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 356 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 siguiente fragmento de código, la cola de clúster se abre con MQOO_BIND_NOT_FIXED y no con 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 278.

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 240.
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 eliminarmensajes 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 valor de *Shareability* y elAtributos de *DefInputOpenOption* . Para aceptar este acceso, utilice la opción MQOO_INPUT_AS_Q_DEF. Consulte la Tabla 35 en la página 226 para ver cómo afecta el valor de estos atributos al tipo de acceso que se le otorgará cuando utilice esta opción.

Tabla 35. 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

Tabla 35. Cómo afectan los atributos de cola y las opciones de la llamada MQOPEN al acceso a las colas (continuación)	
Atributos de colas	**Tipo de acceso con las opciones de MQOPEN**
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 35 en la página 226 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 240.

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 327).

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 39
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" en la página 236
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.

Opción MQOPEN para la desactivación temporal del gestor de colas
En el entorno CICS en z/OS, si utiliza la llamada MQOPEN cuando el gestor de colas está en un estado de inmovilización, la llamada siempre falla.

En otros entornos z/OS , sistemas IBM i, Windows y en entornos de sistemas UNIX and Linux , 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.

Sin embargo, cuando abre una cola remota o una 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 cola remota, o con la cola de clúster remoto elegida.

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. *ResolvedQMgrName* se rellena de forma similar 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 a la que se colocarán los mensajes. *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 231).

Para crear una cola dinámica, utilice una plantilla conocida como cola de modelo, junto con la llamada MQOPEN. Puede crear una cola modelo utilizando los mandatos WebSphere MQ o los paneles de operaciones y 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 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. Además, 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 desea 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 no 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 el campo *ObjectQMgrName* en blanco en este caso.
2. En el campo *ObjectName* de la estructura MQOD, especifique el nombre de la cola remota, tal como lo conoce el gestor de colas *remoto* . En el campo *ObjectQMgrName* , especifique:

   - 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 (no se puede dejar en blanco en este caso).

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 Tabla 34 en la página 222.

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 WebSphere MQ para z/OS, si tiene alguna solicitud MQGET con una opción MQGMO_SET_SIGNAL pendiente para esa cola, se cancelan.
- 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 descriptor de conexión utilizado para abrirlo, o bien, para 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 231
- “Colocación de mensajes en una cola remota” en la página 236
- “Establecimiento de las propiedades de un mensaje” en la página 236
- “Control de la información de contexto” en la página 236
- “Colocar un mensaje en una cola utilizando la llamada MQPUT1” en la página 238
- “Listas de distribución” en la página 240
- “Algunos casos en los que las llamadas put fallan” en la página 245

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

“Obtención de mensajes de una cola” en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

“Confirmación y restitución de unidades de trabajo” en la página 330
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 WebSphere MQ utilizando desencadenantes” en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 355

Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

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 colocando (especificando 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 231
- “Definición de mensajes utilizando la estructura MQMD” en la página 231
- “Especificación de opciones utilizando la estructura MQPMO” en la página 232
- “Los datos del mensaje” en la página 235
- “Transferencia de mensajes: uso de manejadores de mensaje” en la página 236

Especificación de manejadores

Para el descriptor de conexión (*Hconn*) en CICS en aplicaciones z/OS , puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero), o puede utilizar el descriptor 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.

Independientemente del entorno en el que esté trabajando, utilice el mismo manejador de cola (*Hobj*) 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 colocar un nuevo mensaje para asegurarse de que *MsgId* y *CorrelId* son 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 WebSphere MQ" en la página 9, 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 240). 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 coloca tiene información de contexto predeterminada asociada.

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 sí en z/OS; para otras plataformas, es no.

Contexto
Indica el nombre del descriptor de contexto 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 39. 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" en la página 236.

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 (después de 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 240). 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 aparece en la estructura MQOR y utilice MQPMR que contengan campos *MsgId* . Inicialice estos campos *MsgId* en ceros o especifique MQPMO_NEW_MSG_ID.

  Cuando haya colocado el mensaje en la cola, los valores de *MsgId* que el gestor de colas ha creado pasarán a estar disponibles en los MQPMR; puede utilizarlos para identificar qué destino está asociado a cada informe o respuesta.
- Si no desea recibir informes ni respuestas, elija una de las opciones siguientes:
  1. Si desea identificar los destinos que fallan inmediatamente, es posible que desee especificar el mismo valor en el campo *RecsPresent* que aparece en la estructura MQOR y proporcionar MQRR para identificar estos destinos. No especifique ningún MQPMR.
  2. Si no desea identificar los destinos anómalos, 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 244.

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 244 para obtener una visión general de la estructura MQPMR).

Utilice el campo *PutMsgRecPtr* para especificar un puntero al primer registro de colocación de mensajes, o el campo *PutMsgRecOffset* para especificar el desplazamiento del primer registro de colocación 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 243 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 primer 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 en la que está colocando el mensaje
- El tamaño de cualquier cabecera de mensaje añadida por WebSphere MQ (incluida 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 del mensaje que el gestor de colas puede procesar. Tiene un valor predeterminado de 100 MB para todos los productos WebSphere MQ en 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 puede colocar en la cola. Si intenta transferir un mensaje con un tamaño mayor que el valor de este atributo, la llamada MQPUT fallará. Si está colocando un mensaje en una cola remota, el tamaño máximo del mensaje que puede colocar correctamente lo determina el atributo *MaxMsgLength* de la cola remota, de las colas de transmisión intermedias en las que se coloca el mensaje a lo largo de la ruta a su destino y de los canales utilizados.

Para una operación MQPUT, el tamaño del mensaje debe ser menor o igual que el atributo *MaxMsgLength* de la cola y el gestor de colas. Los valores de estos atributos son independientes, pero se recomienda establecer el *MaxMsgLength* de la cola en un valor menor o igual que el del gestor de colas.

WebSphere MQ añade información de cabecera a los mensajes en las circunstancias siguientes:

- Cuando coloca un mensaje en una cola remota, WebSphere 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 WebSphere MQ no puede entregar un mensaje a una cola remota, intenta colocar el mensaje en la cola de mensajes no entregados (undelivered-message). 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, WebSphere MQ añade 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, WebSphere MQ puede añadir 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 los mensajes sea menor que el atributo *MaxMsgLength* de las colas de mensajes no entregados y de transmisión. 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 *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.

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 229. 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 de transmisión, consulte WebSphere MQ técnicas de mensajería distribuida .

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 en un nivel anterior a WebSphere MQ Versión 7.0, que no puede procesar propiedades de mensaje, puede establecer el parámetro PropertyControl en la definición de canal para especificar cómo se van a tratar las propiedades.

Control de la información de contexto

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.

Para controlar la información de contexto, utilice el campo *Options* en la estructura MQPMO.

Si no lo hace, el gestor de colas sobrescribe la información de contexto que ya puede estar en el descriptor de mensaje con la información de identidad y contexto que ha generado para el 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 WebSphere 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 WebSphere 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 referencia de la interfaz de servicios instalables.

Si no desea ninguna información de contexto asociada al mensaje, utilice la opción MQPMO_NO_CONTEXT.

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 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 colocación de mensajes, proporcione el descriptor de contexto de la cola desde la que ha guardado la información de contexto.
- En el campo *Options* de la estructura de opciones de colocación de mensajes, 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 39
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 227
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 221. 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 colocando (especificando 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 aplicaciones CICS , puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero), o utilizar el descriptor de conexión devuelto por 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 desea colocar un mensaje y el nombre del gestor de colas propietario de esta cola.

El campo *DynamicQName* se ignora para la llamada MQPUT1 porque no puede utilizar colas de modelo.

Utilice el campo *AlternateUserId* si desea nombrar un identificador de usuario alternativo que se utilizará para probar la autorización para 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 232.

Cuando el campo *Options* se establece en cero, el gestor de colas utiliza su propio ID de usuario cuando realiza pruebas de autorización para acceder a la cola. Además, el gestor de colas ignora cualquier identificador de usuario alternativo proporcionado 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 soportado en WebSphere MQ para 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.

V 7.5.0.8

Atención: Las listas de distribución no admiten el uso de cola alias que apuntan a objetos de tema. A partir de Version 7.5.0, Fix Pack 8, si una cola alias apunta a un objeto de tema en una lista de distribución, IBM WebSphere 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 un valor mayor que cero en el campo *RecsPresent* , *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 proporciona a través de los registros de objeto (MQOR), que proporcionan detalles de destino (es decir, *ObjectName* y *ObjectQMgrName*).

El descriptor de objeto (*Hobj*) se pasa a la llamada MQPUT, lo que le permite transferir a una lista en lugar de a 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 29 en la página 241, se muestra cómo funcionan las listas de distribución.

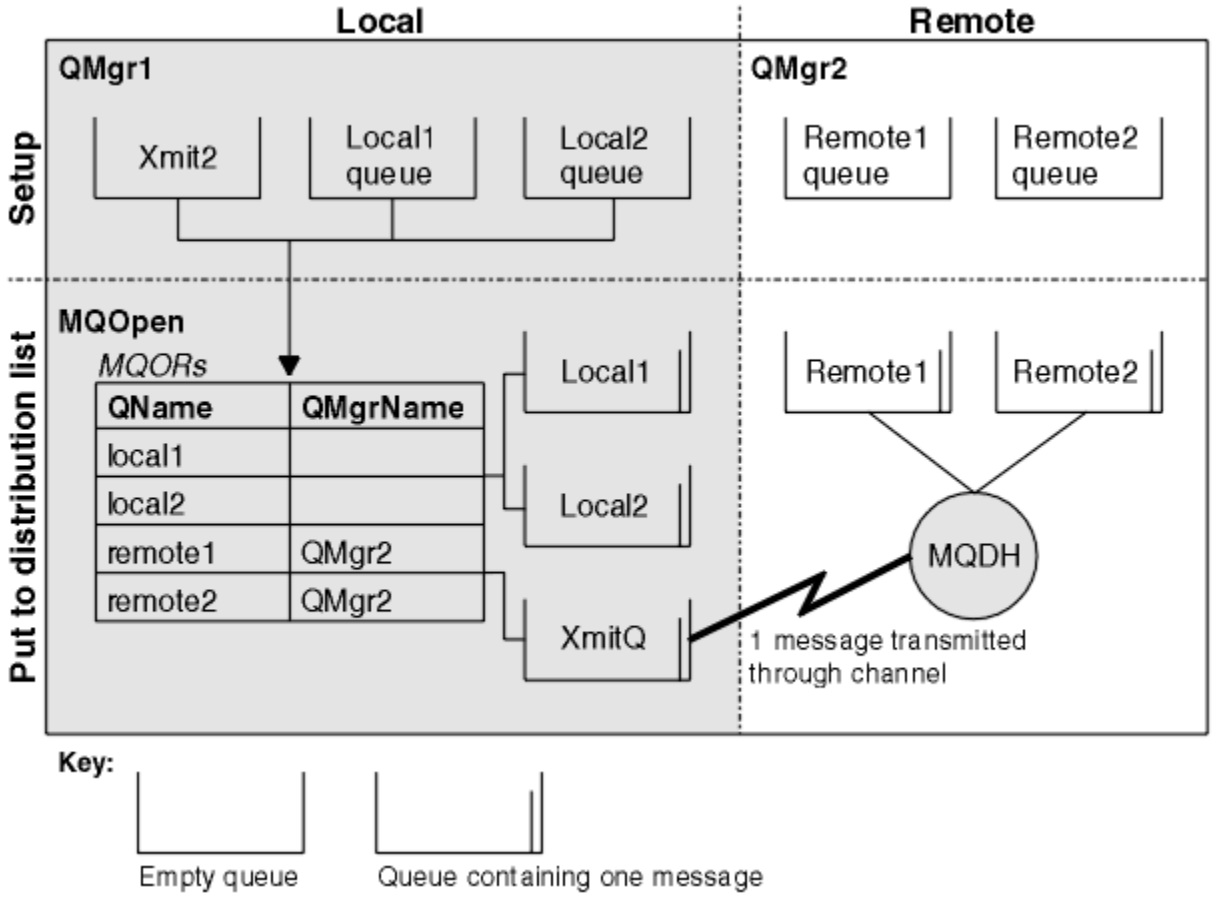


Figura 29. Cómo funcionan las listas de distribución

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 230 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, debe 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 MQOD no se utilizan para las 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).

Cualquiera que sea la técnica que elija, debe utilizar uno de *ObjectRecOffset* y *ObjectRecPtr*; la llamada falla con el código de razón MQRC_OBJECT_RECORDS_ERROR si ambos son cero, o ambos son 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 30 en la página 243 se muestra cómo se puede abrir una lista de distribución en C.

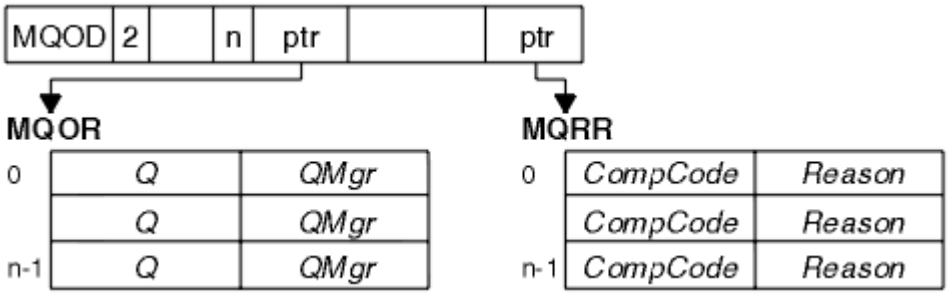


Figura 30. Apertura de una lista de distribución en C

En Figura 31 en la página 243 se muestra cómo se puede abrir una lista de distribución en COBOL.

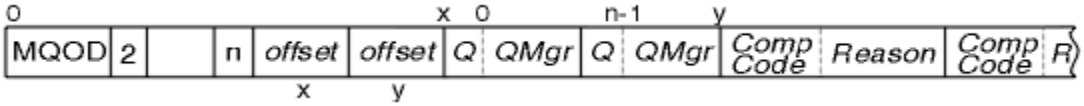


Figura 31. Apertura de una lista de distribución en COBOL

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 219 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 230 para obtener una descripción).
- Un descriptor de objeto. Si se abre una lista de distribución utilizando MQOPEN, el *Hobj* sólo le 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 232 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 131 para obtener una descripción) que muestra el uso de listas de distribución, el ejemplo elige proporcionar valores para *MsgId* y *CorrelId* en 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 32 en la página 244 muestra cómo se puede colocar un mensaje en una lista de distribución en C.

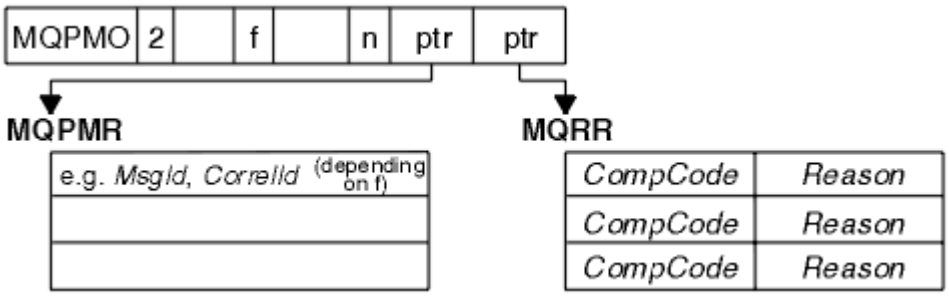


Figura 32. Colocación de un mensaje en una lista de distribución en C

Figura 33 en la página 244 muestra cómo se puede colocar un mensaje en una lista de distribución en COBOL.

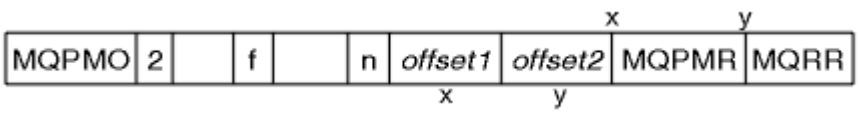


Figura 33. Colocación de un mensaje en una lista de distribución en COBOL

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. Estos no se pueden manejar utilizando sólo los campos de salida *CompCode* y *Reason* en MQOPEN y MQPUT.

Cuando se utilizan listas de distribución para transferir mensajes a varios destinos, los registros de respuesta contienen los *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 211 y “Apertura y cierre de objetos” en la página 219.

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 246
- “Orden en el que se recuperan los mensajes de una cola” en la página 250
- “Obtener un mensaje concreto” en la página 262
- “Mejora del rendimiento de mensajes no persistentes” en la página 264
- “Manejo de mensajes de más de 4 MB de tamaño” en la página 268
- “Espera de mensajes” en la página 274
-
- “Omisión de restitución” en la página 274
- “Conversión de datos de aplicación” en la página 277
- “Cómo examinar mensajes en una cola” en la página 278
- “Algunos casos en los que falla la llamada MQGET” en la página 284

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

“Colocación de mensajes en una cola” en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

“Consulta y establecimiento de atributos de objeto” en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

“Confirmación y restitución de unidades de trabajo” en la página 330
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 WebSphere MQ utilizando desencadenantes” en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

“Cómo trabajar con clústeres y MQI” en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

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 247
- "Descripción de mensajes utilizando la estructura MQMD y la llamada MQGET" en la página 247
- "Especificar opciones MQGET utilizando la estructura MQGMO" en la página 247
- "Especificar el tamaño del área de almacenamiento intermedio" en la página 250

Especificar descriptores de conexión

Para aplicaciones CICS en z/OS , puede especificar la constante MQHC_DEF_HCONN (que tiene el valor cero), o utilizar el descriptor 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.

Utilice el descriptor de contexto de cola (*Hobj*) que se devuelve al llamar a 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 WebSphere MQ" en la página 9, 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 262.

Si no especifica un mensaje concreto, MQGET recupera el *primer* mensaje de la cola. "Orden en el que se recuperan los mensajes de una cola" en la página 250 describe cómo la prioridad de un mensaje, el atributo *MsgDeliverySequence* de la cola y la opción MQGMO_LOGICAL_ORDER determinan el orden de los mensajes en la cola.

Nota: Si desea utilizar MQGET más de una vez (por ejemplo, para recorrer los mensajes de la cola), debe establecer los campos *MsgId* y *CorrelId* de esta estructura en nulo después de cada llamada. De este modo, se borran estos campos de los identificadores del mensaje que se ha recuperado.

Sin embargo, si desea agrupar los mensajes, el *GroupId* debe ser el mismo para los mensajes del mismo grupo, de modo que la llamada busque un mensaje que tenga los mismos identificadores que el mensaje anterior para poder crear todo el grupo.

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 274
- 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
- En WebSphere MQ para z/OS, si el mensaje recuperado está marcado como omisión de restitución (consulte "Omisión de restitución" en la página 274)
- 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
- En WebSphere MQ para z/OS, si se debe permitir 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
- En WebSphere MQ para z/OS, si la llamada falla si la conexión está en un estado de desactivación temporal
- Si es necesaria la conversión de datos del mensaje de aplicación. Consulte "Conversión de datos de aplicación" en la página 277
- El orden en el que los mensajes y (excepto para WebSphere MQ para z/OS) segmentos se recuperan de una cola
- Excepto en WebSphere MQ para z/OS, si los mensajes lógicos completos sólo se pueden recuperar
- Si se pueden recuperar los mensajes de un grupo únicamente cuando están disponibles *todos* los mensajes del grupo
- Excepto en WebSphere MQ para z/OS, si los segmentos de un mensaje lógico sólo se pueden recuperar cuando *todos* los segmentos del mensaje lógico están disponibles

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. Además, en WebSphere MQ para z/OS, la llamada no establece una señal que solicite notificación cuando llega 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 UNIX and Linux	No
z/OS	Sí
Sistemas Windows	No

- En WebSphere MQ para z/OS, el mensaje recuperado no se marca como omisión de 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 tiempo máximo (en milisegundos) que la llamada MQGET espera a que llegue un mensaje a la cola cuando se utiliza la opción MQGMO_WAIT. Si no llega ningún mensaje dentro del tiempo especificado en *WaitInterval*, la llamada se completa y devuelve un código de razón que muestra que no había ningún mensaje que coincidiera con los criterios de selección en la cola.

En WebSphere MQ para 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 274 .

Signal1
Signal1 está soportado en WebSphere MQ para z/OS y MQSeries solo para HP Integrity NonStop Server.

Si utiliza la opción MQGMO_SET_SIGNAL para solicitar que se notifique a la aplicación cuando llegue un mensaje adecuado, especifique el tipo de señal en el campo *Signal1* . En WebSphere MQ en todas las demás plataformas, el campo *Signal1* está reservado y su valor no es significativo.

Signal2
El campo *Signal2* está reservado en todas las plataformas y su valor no es significativo.

ResolvedQName
ResolvedQName es un campo de salida en el que el gestor de colas devuelve el nombre de la cola (después de la resolución de cualquier alias) de la que se ha recuperado el mensaje.

MatchOptions
MatchOptions controla los criterios de selección 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 para el mensaje recuperado.

MsgToken

MsgToken Identifica un mensaje de manera exclusiva.

ReturnedLength
ReturnedLength es un campo de salida en el que el gestor de colas devuelve la longitud de los datos de mensaje devueltos (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 debe contener los datos de mensaje que recupera. 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 la llamada MQGET falla (por ejemplo, porque el almacenamiento intermedio es demasiado pequeño), la longitud del mensaje se devuelve 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. Almacene el *MsgId* de este mensaje y, a continuación, repita la llamada MQGET, especificando un área de almacenamiento intermedio del tamaño correcto y el *MsgId* que ha anotado en la primera llamada.

   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 evitarlo, examine primero la cola hasta que encuentre el mensaje que desee, especificando un *BufferLength* de cero 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 *MaxMsgLength* Cola de determina la longitud máxima de los mensajes aceptados para dicha cola; el atributo *MaxMsgLength* Gestor de colas de determina la longitud máxima de los mensajes aceptados para dicho gestor de colas. Si no sabe qué longitud de mensaje esperar, puede consultar sobre el atributo *MaxMsgLength* (utilizando la llamada MQINQ) y, a continuación, especificar un almacenamiento intermedio 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 sobre el atributo *MaxMsgLength* , consulte “Aumentar la longitud máxima del mensaje” en la página 268.

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.

Priority

Un programa puede asignar una prioridad a un mensaje cuando coloca el mensaje en una cola (consulte “Prioridades de mensajes” en la página 17). 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 más información, consulte “Obtener un mensaje concreto” en la página 262.

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.

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 36. 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 34 en la página 252:

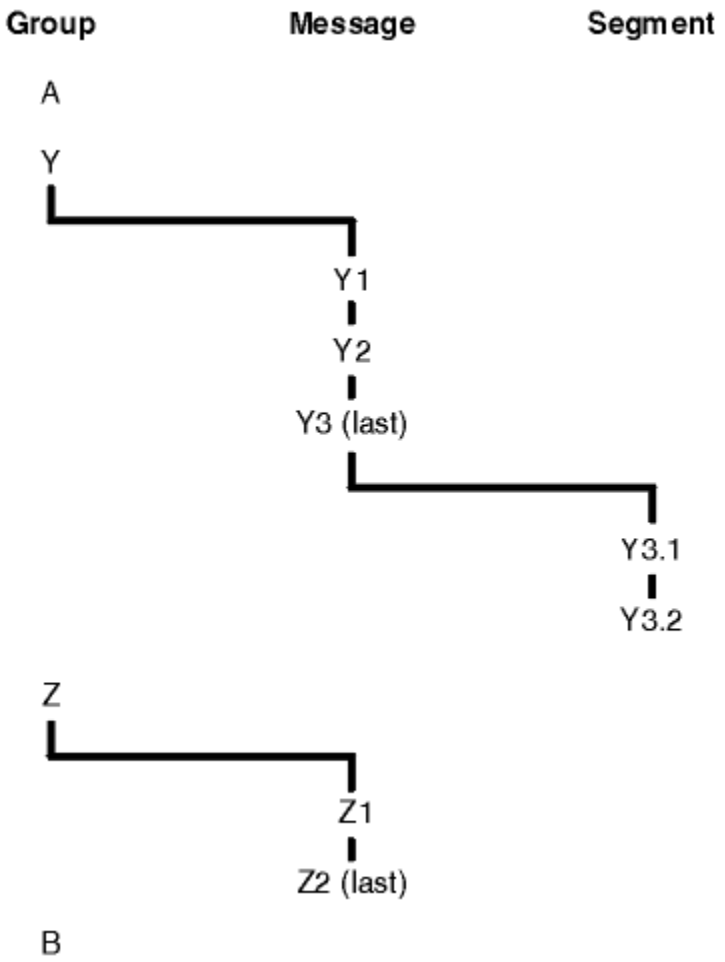


Figura 34. Orden lógico en una cola

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 35 en la página 253:

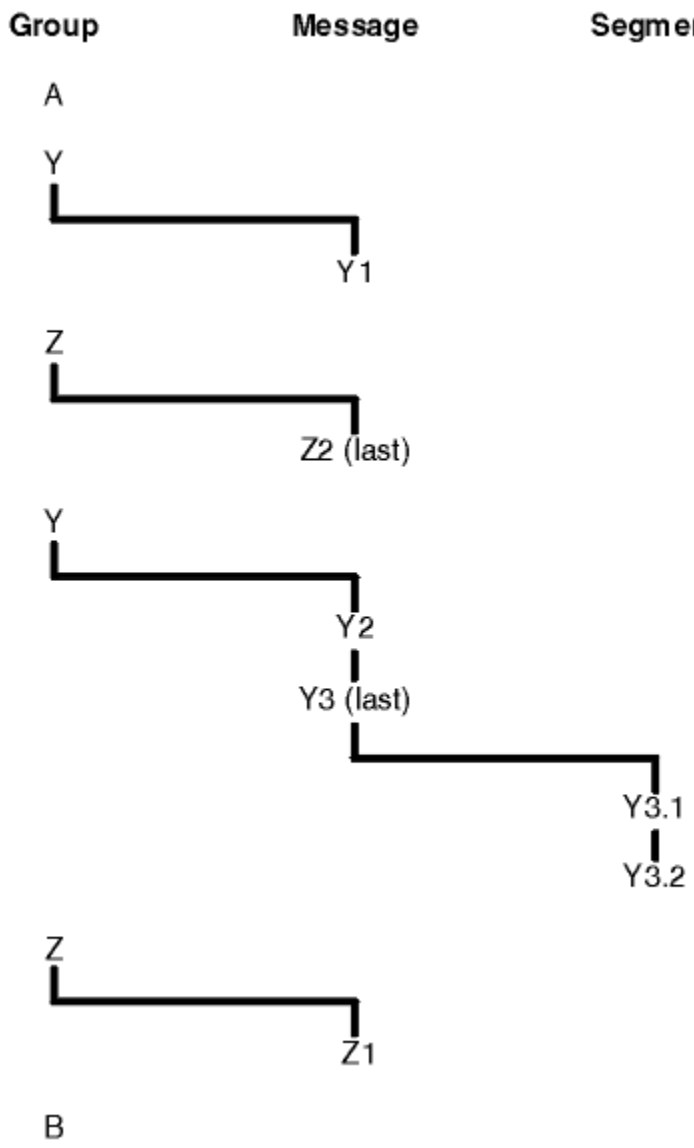


Figura 35. Orden físico en una cola

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 WebSphere 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 del grupo en la cola, utilizando MQGMO_BROWSE_NEXT para examinar más allá del final del grupo, especificando MQMO_MATCH_MSG_SEQ_NUMBER con *MsgSeqNumber* establecido en 1 (para buscar el primer mensaje del siguiente grupo) 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 WebSphere MQ incrementa automáticamente el número de secuencia de mensajes.
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.

Después de que se haya iniciado un grupo de mensajes o un mensaje lógico, las llamadas MQPUT posteriores deben especificar los distintivos MQMF_ * adecuados 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.

La Figura 35 en la página 253 muestra las combinaciones de opciones y distintivos que son válidos, y los valores de los campos *GroupId*, *MsgSeqNumber*y *Offset* que el gestor de colas utiliza 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 36. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos

Opciones especificadas				**Estado de grupo y mensaje lógico antes de la llamada**		**Valores que utiliza el gestor de colas**		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***
Sí	No	No	No	No	No	MQGI_NONE	1	0
Sí	No	No	Sí	No	No	Nuevo ID de grupo	1	0
Sí	No	Sí	Cualquiera de los dos	No	No	Nuevo ID de grupo	1	0

Tabla 36. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos (continuación)

Opciones especificadas				**Estado de grupo y mensaje lógico antes de la llamada**		**Valores que utiliza el gestor de colas**		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***
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
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

Tabla 36. Opciones de MQPUT relacionadas con los mensajes de grupos y los segmentos de mensajes lógicos (continuación)

Opciones especificadas				Estado de grupo y mensaje lógico antes de la llamada		Valores que utiliza el gestor de colas		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***

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 y, de lo contrario, utiliza el valor del campo.
- Para el campo *CorrelId* , el gestor de colas genera un nuevo identificador de correlación si se especifica MQPMO_NEW_CORREL_ID y, de lo contrario, utiliza el valor del 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. Es responsabilidad de la aplicación asegurarse de que los campos *GroupId*, *MsgSeqNumber*, *Offset*y *MsgFlags* tienen los valores adecuados.

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 adecuados y, a continuación, emitir la llamada MQPUT con MQPMO_SYNCPOINT o MQPMO_NO_SYNCPOINT establecido según sea necesario, 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.Tabla 37 en la página 258 muestra los diversos casos que pueden surgir. 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 37. 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 36.

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 271 y "Colocación y obtención de un grupo que abarca varias unidades de trabajo" en la página 260.

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 este caso, StatusInfo debe incluir *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 la segmentación está implicada (en este caso, StatusInfo debe incluir *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 determinado 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 38 en la página 263 muestra qué mensaje se recupera para los valores posibles de estos campos. Estos campos se ignoran en la entrada si especifica MQGMO_MSG_UNDER_CURSOR en el parámetro *GetMsgOpts* de la llamada MQGET.

Tabla 38. 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.

Cuando se devuelve, 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* . Tabla 39 en la página 263 muestra qué mensaje se recupera para los posibles valores de estos campos.

Tabla 39. 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 que debe coincidir.
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 juntos para proporcionar el segmento identificado por 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 251 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 utiliza 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 restablecer los campos *MsgId* o *CorrelId* entre MQGETs.

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 22.

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 llama a MQOPEN con MQOO_READ_AHEAD, el cliente de WebSphere MQ sólo habilita la lectura anticipada si se cumplen determinadas condiciones. Estas condiciones incluyen:

- El cliente y el gestor de colas remoto deben ser de WebSphere MQ Versión 7 o posterior.
- La aplicación cliente debe compilarse y enlazarse con las bibliotecas de cliente MQI WebSphere 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 de 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 suscriptora solicita que WebSphere MQ gestione el destino al que se envían las publicaciones, se crea una cola gestionada como una cola dinámica basada en una cola 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 para mensajes no persistentes en AIX" en la página 267
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 266
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 265
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 llama a MQOPEN con MQOO_READ_AHEAD, el cliente de WebSphere MQ sólo habilita la lectura anticipada si se cumplen determinadas condiciones. Estas condiciones incluyen:

- El cliente y el gestor de colas remoto deben ser de WebSphere MQ Versión 7 o posterior.
- La aplicación cliente debe compilarse y enlazarse con las bibliotecas de cliente MQI WebSphere 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 40. Opciones de MQGET y lectura anticipada

	Se permite cuando la lectura anticipada está habilitada y se puede modificar entre llamadas MQGET[5]	**Permitidas cuando la lectura anticipada está habilitada, pero no se pueden modificar entre llamadas MQGET[1]**	**Opciones MQGET que no están permitidas cuando la lectura anticipada está habilitada[2]**
Valores de MQGET MQMD	MsgId[3] CorrelId[3]	Codificación CodedCharSetId	

Tabla 40. Opciones de MQGET y lectura anticipada (continuación)			
	Se permite cuando la lectura anticipada está habilitada y se puede modificar entre llamadas MQGET[5]	**Permitidas cuando la lectura anticipada está habilitada, pero no se pueden modificar entre llamadas MQGET[1]**	**Opciones MQGET que no están permitidas cuando la lectura anticipada está habilitada[2]**
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 llama a MQOPEN con MQOO_READ_AHEAD, el cliente de WebSphere MQ sólo habilita la lectura anticipada si se cumplen determinadas condiciones. Estas condiciones incluyen:

- El cliente y el gestor de colas remoto deben ser de WebSphere MQ Versión 7 o posterior.
- La aplicación cliente debe compilarse y enlazarse con las bibliotecas de cliente MQI WebSphere 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 para 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 le proporciona el IBM WebSphere MQ `MQRC_UNEXPECTED_ERROR (2195)` y, si hay suficientes descriptores de archivo, un archivo IBM WebSphere 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
```

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, WebSphere MQ proporciona una serie de formas de tratar los mensajes que tienen una longitud superior a 4 MB.

Puede aumentar el atributo *MaxMsgLength* hasta 100 MB en todos los sistemas WebSphere MQ en V6 o posterior. Establezca este valor para que refleje el tamaño de los mensajes que usan la cola. En sistemas WebSphere MQ que no sean WebSphere 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 de gestor de colas *MaxMsgLength* define la longitud máxima de un mensaje que puede manejar un gestor de colas. De forma similar, el atributo de cola *MaxMsgLength* es la longitud máxima de un mensaje que una cola puede manejar. La longitud máxima de mensaje *default* soportada depende del entorno en el que esté trabajando.

Si está manejando mensajes de gran tamaño, puede modificar estos atributos de forma independiente. Puede establecer el valor del atributo del gestor de cola en el rango de 32768 bytes a 100 MB; puede establecer el valor del atributo de cola en el rango de 0 a 100 MB.

Después de cambiar uno o ambos atributos de *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 menor o igual que los atributos *MaxMsgLength* de la cola y 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 lo determina el *MaxMsgLength* para cada cola (incluidas las colas de transmisión) en la que se colocará el mensaje. Este valor suele tomar como valor predeterminado 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.

Segmentación de mensajes

Utilice esta información para obtener información acerca de la segmentación de mensajes.

Nota: No soportado en IBM WebSphere MQ para z/OS o por aplicaciones que utilizan clases IBM WebSphere MQ para 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 268 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 36.

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).

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 260.

Sin embargo, no puede poner o recibir mensajes segmentados en una unidad de trabajo global.

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
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.

Segmentación de aplicaciones
La segmentación de aplicaciones se utiliza cuando la segmentación del gestor de colas no es adecuada, o cuando las aplicaciones requieren la 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 que las aplicaciones puedan manejarlo en un único almacenamiento intermedio.
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 siempre utiliza MQGMO_COMPLETE_MSG, la segmentación del gestor de colas también se puede permitir 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 utiliza MQPMO_LOGICAL_ORDER, la aplicación debe establecer *Offset* 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
```

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 soportado en WebSphere MQ para z/OS.

Este método permite transferir un objeto grande de un nodo a otro sin almacenar el objeto en colas de WebSphere MQ 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 420.

WebSphere 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 bajo un ID de usuario de WebSphere 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 masivos añadidos al mismo debe viajar a través de varios gestores de colas antes de llegar a su destino, los datos masivos *están* presentes en las colas de WebSphere MQ en los nodos intermedios. 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 WebSphere 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 sea necesario 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 MQHREF, cuando se solicita en MQGET, se convierte automáticamente mediante WebSphere MQ junto con los datos masivos siguientes.

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 todo el objeto)
- *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 inicio 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 “Programas de ejemplo para plataformas distribuidas” en la página 98.

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 especificartiempo 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 utilizando 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 IMS no deben especificar un intervalo de espera ilimitado porque esto impediría que el sistema IMS terminara. (Cuando IMS termina, requiere 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 después de 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.
- 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.

Para obtener más información sobre las circunstancias en las que se producen estas acciones, consulte MQGMO.

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 se admite en WebSphere MQ para 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 WebSphere MQ para confirmar la nueva unidad de trabajo o para 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 36 en la página 276 ilustra una secuencia típica de pasos que puede contener un programa de aplicación cuando una solicitud MQGET debe omitir la restitución.

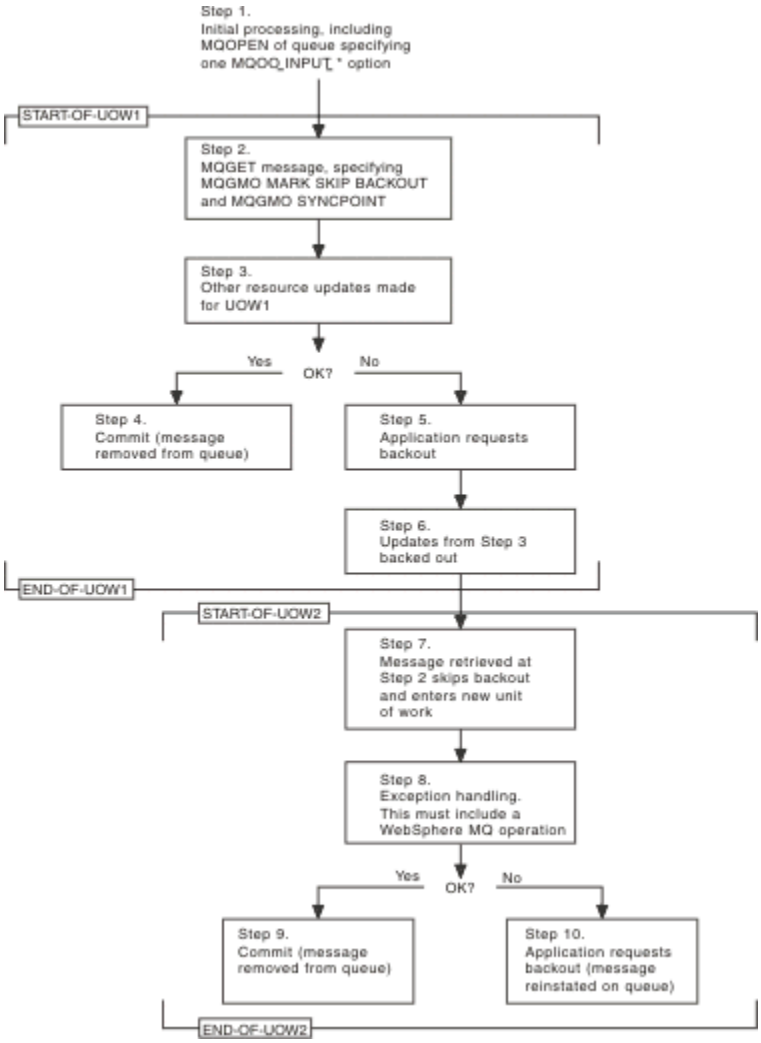


Figura 36. Omisión de restitución utilizando MQGMO_MARK_SKIP_BACKOUT

Los pasos de la Figura 36 en la página 276 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 36 en la página 276, 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 de aplicación se convierten dentro de un programa de aplicación cuando se especifica la opción MQGMO_CONVERT en el campo *Options* de la estructura MQGMO pasada a una llamada MQGET, y *todo* lo siguiente es cierto:

- Los campos *CodedCharSetId* o *Encoding* establecidos en la estructura MQMD asociada con el mensaje en 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 con el mensaje no es MQFMT_NONE.
- El *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 las estructuras MQMD asociadas con el 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 con el mensaje es uno de los formatos incorporados, el gestor de colas puede convertir el mensaje. Si *Format* no es uno de los formatos incorporados, debe escribir una salida de conversión de datos para 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 en el gestor de colas emisor y el campo *Feedback* de la estructura MQDLH indica la razón. 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 424. 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 fuerce a WebSphere MQ a utilizar las tablas de conversión no modificadas, pero debe tener en cuenta el comportamiento incoherente que puede resultar.

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 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 el que se copian los mensajes de la cola viene 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 250).

- "Colas FIFO (primero en entrar, primero en salir)" en la página 279
- "Colas en secuencia de prioridad" en la página 279
- "Mensajes sin confirmar" en la página 279
- "Cambio de la secuencia de una cola" en la página 280
- "Uso de un índice de cola" en la página 280

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 se admite en WebSphere MQ para 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 examinar un mensaje cuando no conoce 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 *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, vuelva a llamar a MQGET, 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 251 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 281, 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

Empiece a escribir aplicaciones de publicación/suscripción de WebSphere MQ .

Para obtener una visión general de los conceptos de publicación/suscripción, consulte Introducción a la mensajería de publicación/suscripción de WebSphere MQ.

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 285
- “Escritura de aplicaciones de suscriptor” en la página 292
- “Ciclos de vida de publicación/suscripción” en la página 310
- “Propiedades de los mensajes de publicación/suscripción” en la página 315
- “Orden de los mensajes” en la página 317
- “Interceptación de publicaciones” en la página 317
- “Opciones de publicación” en la página 325
- “Opciones de suscripción” en la página 325

Conceptos relacionados

"Conceptos de desarrollo de aplicaciones" en la página 8
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

"Decidir qué lenguaje de programación utilizar" en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

"Diseño de aplicaciones IBM WebSphere MQ" en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

"Programas WebSphere MQ de ejemplo" en la página 98
Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

"Escritura de una aplicación de gestión de colas" en la página 199
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 cliente" en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

"Utilización de servicios web en WebSphere MQ" en la página 965
Puede desarrollar aplicaciones IBM WebSphere MQ para servicios web utilizando el transporte IBM WebSphere MQ para SOAP o el puente IBM WebSphere MQ para HTTP.

"Creación de una aplicación IBM WebSphere MQ" en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

"Manejo de errores de programa" en la página 559
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.

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 de WebSphere MQ simple es como escribir una aplicación de punto a punto de WebSphere MQ que transfiere mensajes a una cola (Tabla 41 en la página 285). La diferencia es que envía mensajes MQPUT a un tema, no a una cola.

Tabla 41. Punto a punto frente al patrón de programa WebSphere MQ 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	

Tabla 41. Punto a punto frente al patrón de programa WebSphere MQ de publicación/suscripción. (continuación)

Paso	Llamada MQ punto a punto	Llamada MQ de publicación
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 286), 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 289), el patrón de interacción del programa con WebSphere 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 292
Empiece con la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación WebSphere MQ que consume mensajes de una cola, una aplicación que crea una suscripción y no requiere conocimientos de colas y, por último, un ejemplo que utiliza tanto colas como suscripciones.

Referencia relacionada

DEFINE TOPIC
DISPLAY TOPIC
DISPLAY TPSTATUS

Ejemplo 1: Publicador en un tema fijo

Un programa de WebSphere MQ para ilustrar 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 38 en la página 287

```
#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 37. Publicador de WebSphere MQ simple para 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 38. Salida de ejemplo del primer ejemplo de publicador

Las siguientes líneas de código seleccionadas ilustran aspectos de la escritura de una aplicación de publicación para WebSphere 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 de antemano, utilizando WebSphere 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 WebSphere 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 nulo a WebSphere MQ como parte del almacenamiento intermedio de mensajes.

¿Qué demuestra el primer ejemplo? El ejemplo imita lo más cerca posible el patrón tradicional probado y probado para escribir programas de punto a punto de WebSphere MQ . Una característica importante del patrón de programación WebSphere MQ es que el programador no está preocupado 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. Incorporado en la arquitectura del modelo de publicación/suscripción de WebSphere MQ es la noción de un objeto de tema administrativo que proporciona la parte raíz, o 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 289 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 289
Programa de WebSphere MQ que muestra la publicación en un tema definido por programa.

“Escritura de aplicaciones de suscriptor” en la página 292
Empiece con la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación WebSphere MQ que consume mensajes de una cola, una aplicación que crea una suscripción y no requiere conocimientos de colas y, por último, 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 40 en la página 291.

```
#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 39. Publicador de WebSphere MQ simple para 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 40. 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 un "/" 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. WebSphere 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 41 en la página 292). 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 hay ningún nombre de tema y el programa proporciona la serie de tema por completo. El resultado que se muestra en la Figura 40 en la página 291 ilustra cómo el programa proporciona dinámicamente la serie de tema completa.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
Para el caso predeterminado, es necesario crear el topicName opcional de antemano, utilizando WebSphere MQ Explorer o este mandato 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

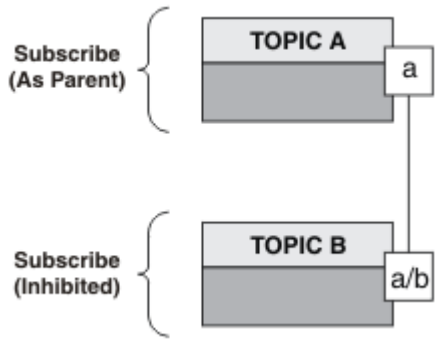


Figura 41. Asociaciones de objeto de tema

¿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. Para la publicación/suscripción, no lo hacen, aunque WebSphere MQ implementa la publicación/suscripción utilizando su sistema de colas subyacente; las aplicaciones de publicación/suscripción heredan las ventajas de la entrega garantizada, la transaccionalidad y el acoplamiento flexible asociados con la mensajería y la colocación en colas.

Un diseñador tiene que decidir si el publicador, y el suscriptor, los programas deben ser conscientes del árbol de temas subyacente o no, y también si los programas del suscriptor son conscientes de poner en cola 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 286
Un programa de WebSphere MQ para ilustrar la publicación en un tema definido administrativamente.

"Escritura de aplicaciones de suscriptor" en la página 292
Empiece con la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación WebSphere MQ que consume mensajes de una cola, una aplicación que crea una suscripción y no requiere conocimientos de colas y, por último, un ejemplo que utiliza tanto colas como suscripciones.

Escritura de aplicaciones de suscriptor

Empiece con la escritura de aplicaciones de suscriptor estudiando tres ejemplos: una aplicación WebSphere MQ que consume mensajes de una cola, una aplicación que crea una suscripción y no requiere conocimientos de colas y, por último, un ejemplo que utiliza tanto colas como suscripciones.

En Tabla 42 en la página 293 se listan los tres estilos de consumidor o suscriptor, junto con las secuencias de llamadas de función de WebSphere 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 hace que las publicaciones se direccionen a la cola se crea administrativamente utilizando WebSphere 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.
3. En el tercer estilo, la aplicación de suscriptor elige abrir y cerrar la cola subyacente que se utiliza para las publicaciones, así como emitir suscripciones para rellenar la cola con publicaciones.

Una forma de entender estos estilos es estudiar los programas C de ejemplo listados en Tabla 42 en la página 293 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 285.

Tabla 42. Punto a punto frente a los patrones de programa de suscripción de WebSphere MQ .

Paso	Consumidor de mensajes MQ	"Ejemplo 1: consumidor de Publicación MQ" en la página 293	"Ejemplo 2: Suscriptor gestionado de MQ" en la página 296	"Ejemplo 3: Suscriptor MQ no gestionado" en la página 301
Conectarse a un gestor de colas	MQCONN	MQCONN	MQCONN	MQCONN
Abrir cola	MQOPEN	MQOPEN		MQOPEN
Suscribir			MQSUB	MQSUB
Obtener mensajes	MQGET	MQGET	MQGET	MQGET
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. Puesto que es poco probable que tenga que especificar las opciones MQCLOSE cuando la cola de suscripción se cierra en el caso del suscriptor MQ gestionado y el argumento de simetría no es relevante, la cola de suscripción no se cierra explícitamente en el 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 310 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 WebSphere MQ que no se suscribe a sí mismo a los temas.

Para crear la cola de suscripción y publicación para este ejemplo, ejecute los mandatos siguientes o defina los objetos utilizando WebSphere 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 301

Nota: El estilo de codificación compacto está pensado para facilitar la lectura, no para su uso en producción.

Los resultados se muestran en Figura 43 en la página 295.

```
#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 42. 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 43. Salida del consumidor de Publicación MQ

Hay un par de sugerencias de programación de lenguaje estándar de WebSphere MQ *C* 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 WebSphere 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 valor nulo al final del almacenamiento intermedio de mensajes para asegurarse de que el mensaje devuelto tiene un valor nulo final en caso de que `"if (messlen == strlen(publication));"` sea verdadero. 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 gestionado de MQ” en la página 296
El suscriptor MQ gestionado es el patrón preferido para la mayoría de aplicaciones de suscriptor. El ejemplo no requiere *ninguna* definición administrativa de colas, temas o suscripciones.

“Ejemplo 3: Suscriptor MQ no gestionado” en la página 301
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. El ejemplo muestra distintas formas de combinar suscripciones y colas.

“Escribir aplicaciones de publicación” en la página 285
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 gestionado de MQ

El suscriptor MQ gestionado es el patrón preferido para la mayoría de aplicaciones de suscriptor. El ejemplo no requiere *ninguna* definición administrativa de colas, temas o suscripciones.

Este tipo de más simple de suscriptor gestionado por suele utilizar una suscripción *no duradera* . El ejemplo está pensado para una suscripción no duradera. La suscripción solo dura tanto como el tiempo de vida del descriptor de contexto de suscripción de MQSUB. Las publicaciones Cualquiera que coincidan con la serie de tema durante el tiempo de vida de la suscripción se envían a la cola de suscripción (y posiblemente una publicación retenida si el distintivo `MQSO_NEW_PUBLICATIONS_ONLY` no está establecido o no es el valor predeterminado, se ha retenido una publicación anterior que coincidía con la serie de tema y la publicación era persistente o el gestor de colas no ha terminado, desde que se creó la publicación).

También puede utilizar una suscripción *duradera* con este patrón. Normalmente si se utiliza una suscripción duradera gestionada, se realiza por motivos de fiabilidad, en lugar de establecer una suscripción que, sin que se produzcan errores, sobreviviría al suscriptor. Para obtener más información sobre los diferentes ciclos de vida asociados con 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 de : 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 de autogestión: las suscripciones *no-duraderas* solo existen mientras dura la aplicación de suscriptor.
   - Si define una suscripción gestionada *duradera* , da como resultado una cola de suscripción permanente y las publicaciones se siguen almacenando en ella sin ningún programa de suscriptor que 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 establecer `SubExpiry` para suscripciones duraderas para que las publicaciones dejen de enviarse a la cola y el suscriptor pueda consumir las publicaciones restantes antes de eliminar la suscripción y hacer que el gestor de colas suprima la cola y las publicaciones restantes de 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, una aplicación se puede desplegar sin que la aplicación cree accidentalmente un árbol de temas que se solapa con otro árbol de temas creado por otra instancia u otra aplicación.
5. Temas administrados: utilizando una serie de tema en la que la primera parte coincide con un objeto de tema definido administrativamente, las publicaciones se gestionan de acuerdo con los atributos del objeto de tema.
   - Por ejemplo, , si la primera parte de la serie de tema coincide con la serie de tema asociada con 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 "sub-topics" sin preocuparse por la gestión de temas.
   - es la serie de tema resultante que se utiliza para coincidir con el objeto de tema que proporciona los atributos asociados con el tema, y no necesariamente el objeto de tema denominado en `sd.Objectname`, aunque normalmente resultan ser uno y el mismo. Consulte “Ejemplo 2: aplicación de publicación en un tema variable” en la página 289.

Al hacer que la suscripción sea duradera en el ejemplo, las publicaciones se siguen enviando a la cola de suscripción después de que el suscriptor haya cerrado 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 temporalmente los valores predeterminados proporcionando sus propias colas de modelo duraderas y no duraderas en cada tema 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 entradaexclusiva.

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 siguiente ejemplo, “Ejemplo 3: Suscriptor MQ no gestionado” en la página 301, 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 46 en la página 299.

```
#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 44. Suscriptor gestionado de MQ -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 explícitamente una cola de suscripción gestionada no duradera para recibir publicaciones, pero sí necesita asignar almacenamiento para el descriptor de contexto de objeto que devuelve el gestor de colas cuando abre la cola. Es importante inicializar el descriptor de contexto en MQHO_OBJECT. Esto indica al gestor de colas que necesita devolver un descriptor de contexto de cola a la cola de suscripción.

MQSD sd = {MQSD_DEFAULT};
El nuevo descriptor de suscripción utilizado en MQSUB.

MQCHAR48 qName;
Aunque el ejemplo non' t requiere conocimiento de la cola de suscripción, el ejemplo sí consulta el nombre de la cola de suscripción-el enlace MQINQ es un poco incómodo en el lenguaje C, por lo que puede encontrar 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 45. Suscriptor de MQ gestionado-parte 2: cuerpo de 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 46. Salida del suscriptor gestionado de MQ

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, el ejemplo especifica sólo las que se utilizan con más frecuencia; las otras se dejan en default.

1. Como la suscripción es *no duradera*, es decir, tiene un tiempo de vida de la suscripción abierta en la aplicación, establezca el MQSO_CREATE distintivo. También puede establecer el distintivo *(predeterminado)* MQSO_NON_DURABLE para la legibilidad.
2. MQSO_CREATE se complementa con MQSO_RESUME. Ambos distintivos se pueden establecer juntos; el gestor de colas crea una suscripción nueva o reanuda una suscripción existente, según corresponda. 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. Dados los permisos correctos, las propiedades de una suscripción reanudada se cambian para que coincidan con 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 301.
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 WebSphere MQ para devolver 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. Esto impide sobrescribir el terminador nulo que se proporciona, en caso de que la serie de tema resuelta llene todo el almacenamiento intermedio.

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 no es duradero, probablemente no esté interesado en su nombre, aunque puede inspeccionar su estado en WebSphere MQ Explorer. Puede proporcionar el

parámetro sd.SubName como input, para que sepa qué nombre buscar; obviamente, 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 293
El consumidor de Publicación MQ es un consumidor de mensajes de IBM WebSphere MQ que no se suscribe a sí mismo a los temas.

"Ejemplo 3: Suscriptor MQ no gestionado" en la página 301
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. El ejemplo muestra distintas formas de combinar suscripciones y colas.

"Escribir aplicaciones de publicación" en la página 285
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. 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.

El gestor de colas abre implícitamente la cola de suscripciones gestionadas duraderas para un suscriptor de tal manera que el proceso compartido de la cola no es posible. Además, no puede crear más de una suscripción para cada cola gestionada y es posible que encuentre las colas más difíciles de gestionar 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 49 en la página 307 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. Todas las suscripciones se visualizan en WebSphere 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, al codificar una opción específica de `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` , el gestor de colas devuelve un error (Tabla 43 en la página 303) si los estados de la suscripción y la cola proporcionados como entrada a MQSUB son incoherentes con el valor de `sd.Options`. Figura 55 en la página 310 muestra los resultados de emitir MQSUB para la suscripción X con distintos valores individuales del distintivo `sd.Options` y pasarle tres manejadores de objeto diferentes.

  Explore las diferentes entradas para el programa de ejemplo en la Figura 48 en la página 306 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. normalmente se debe a que se pasa 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. Tenga cuidado con esta capacidad, ya que puede llevar a "solapar" suscripciones y 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. Es una variación del suscriptor de MQ no gestionado, pero en lugar de emitir MQOPEN y MQSUB en el mismo programa, un programa se suscribe a las 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 43 en la página 303 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 50 en la página 308 a la Figura 55 en la página 310.

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 43 en la página 303, `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 pasados a `MQSUB` difieren de los atributos de la suscripción, `MQSO_RESUME` falla, mientras que `MQSO_ALTER` se ejecuta correctamente siempre que la instancia de programa tenga permiso para modificar los atributos. Tenga en cuenta que nunca puede cambiar la serie de tema en una suscripción; pero en lugar de devolver un error, `MQSUB` ignora el nombre de tema y los valores de serie de tema en el descriptor de suscripción y utiliza los valores de la suscripción existente.

A continuación, consulte Tabla 43 en la página 303 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 porque se trata de 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 descriptor de contexto de objeto 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`. Esta llamada falla con un *RC = 2428-la suscripción X no existe en la cola A* .

Tabla 43. Errores de MQSUB con diferentes combinaciones de suscripciones y 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

Tabla 43. Errores de MQSUB con diferentes combinaciones de suscripciones y descriptores de contexto de cola (continuación)

	Cola A **Suscripción X** **Cola B** **Ninguna suscripción**	**Cola A** **Ninguna suscripción** **Cola B** **Ninguna suscripción**
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 47. 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 48. 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]))
Tiene la opción de especificar Alter | C reate | Resume en el parámetro 5, para probar el efecto de alterar temporalmente parte del valor de opción MQSUB utilizado de forma predeterminada en el ejemplo. El valor predeterminado utilizado por 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 se debe establecer y hacer referencia a una suscripción que se puede reanudar o modificar.

***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 nula, 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 49. 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 suscripción, el ejemplo utiliza MQHO_NONE como 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. En función de si la suscripción es duradera o no, la suscripción se cierra con un MQCO_KEEP_SUB o MQCO_REMOVE_SUBimplícito. 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 que detiene cualquier publicación adicional que se envíe a la cola de suscripción.

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 Figura 50 en la página 308 , el ejemplo se inicia publicando 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 50. Publicar 130 en NYSE/IBM/PRICE

En la ejecución Figura 51 en la página 308 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 55 en la página 310. 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 opción o combinación de MQSO_CREATE, MQSO_RESUMEy 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 51. Recibir la publicación retenida

En (Figura 52 en la página 309) no se reciben publicaciones, porque 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 se debe a que el MQOPEN implícito de STOCKTICKER 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 52. Reanudar la suscripción

En la Figura 53 en la página 309, 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 53. Recibir la publicación retenida con una nueva suscripción no gestionada no duradera

En la Figura 54 en la página 309, 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 54. Solapamiento de suscripciones

En la Figura 55 en la página 310, 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 55. Los temas de suscripción no se pueden cambiar

Conceptos relacionados

“Ejemplo 1: consumidor de Publicación MQ” en la página 293
El consumidor de Publicación MQ es un consumidor de mensajes de IBM WebSphere MQ que no se suscribe a sí mismo a los temas.

“Ejemplo 2: Suscriptor gestionado de MQ” en la página 296
El suscriptor MQ gestionado es el patrón preferido para la mayoría de aplicaciones de suscriptor. El ejemplo no requiere *ninguna* definición administrativa de colas, temas o suscripciones.

“Escribir aplicaciones de publicación” en la página 285
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 de WebSphere MQ como las colas se crean administrativamente, o mediante programas administrativos que utilizan 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[2]. Para

[2] El editor debe enviar los registros de pasajeros como mensajes persistentes para evitar otros posibles fallos, por supuesto.

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 56 en la página 312 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.

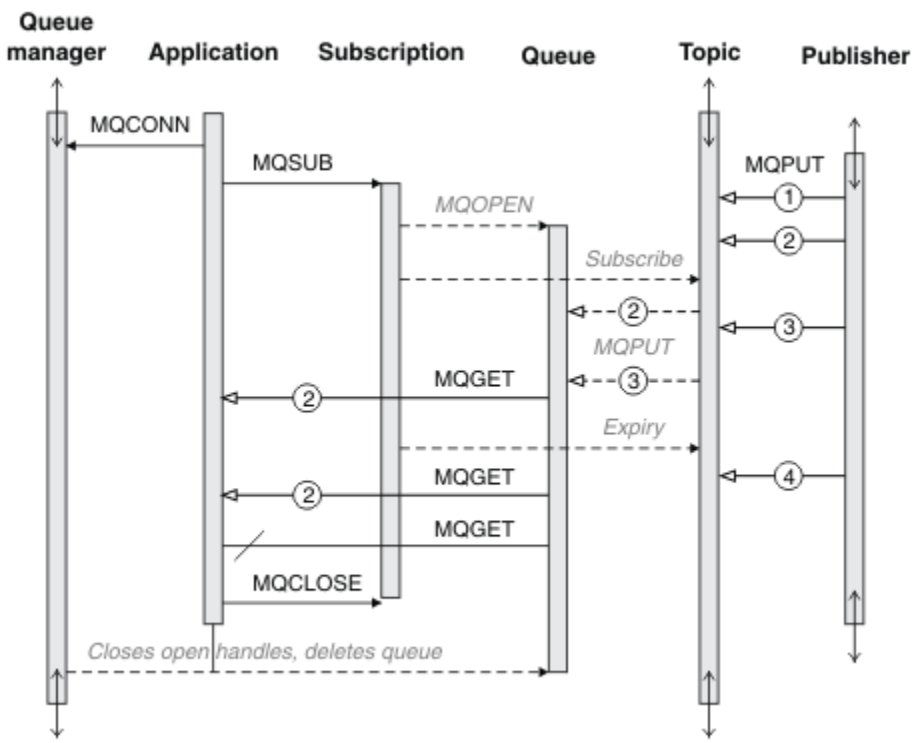


Figura 56. Líneas de vida de un suscriptor gestionado no duradero

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_DELETEo MQCO_PURGE_DELETE. Los detalles de implementación visibles de una suscripción gestionada no forman parte de la interfaz de WebSphere MQ soportada. El gestor de colas no puede gestionar una suscripción de forma fiable a menos que tenga un control completo.

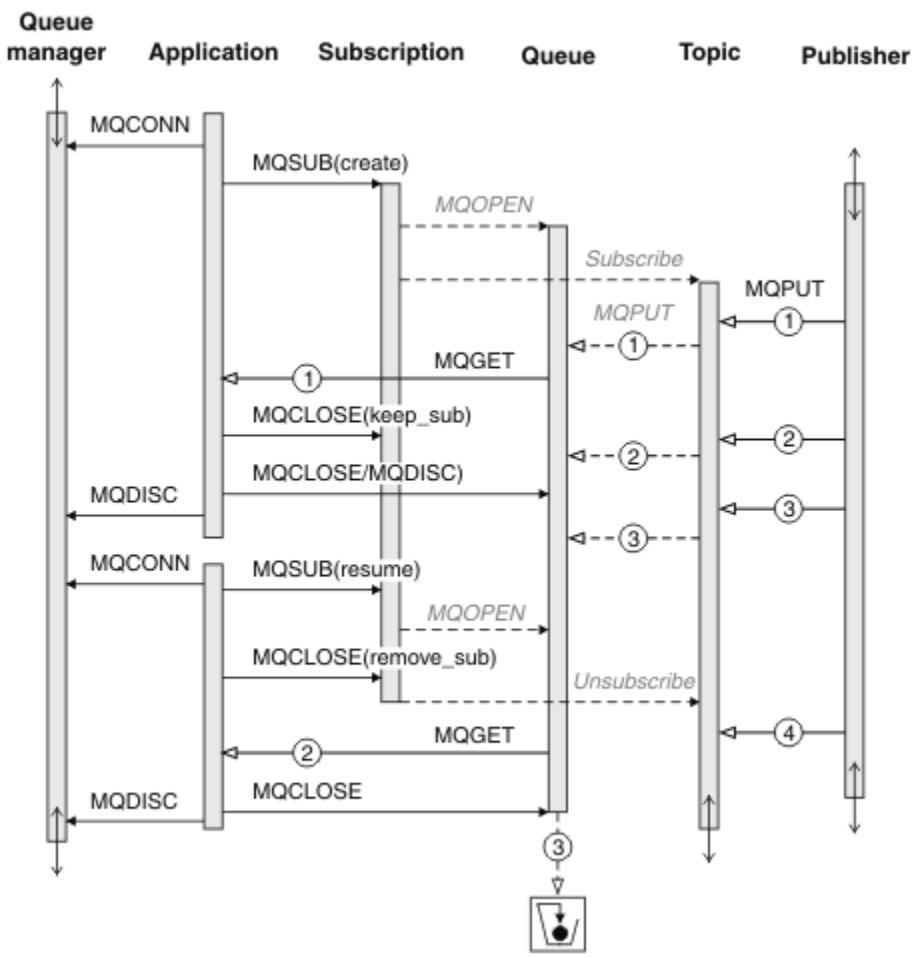


Figura 57. Líneas de vida de un suscriptor duradero gestionado

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.

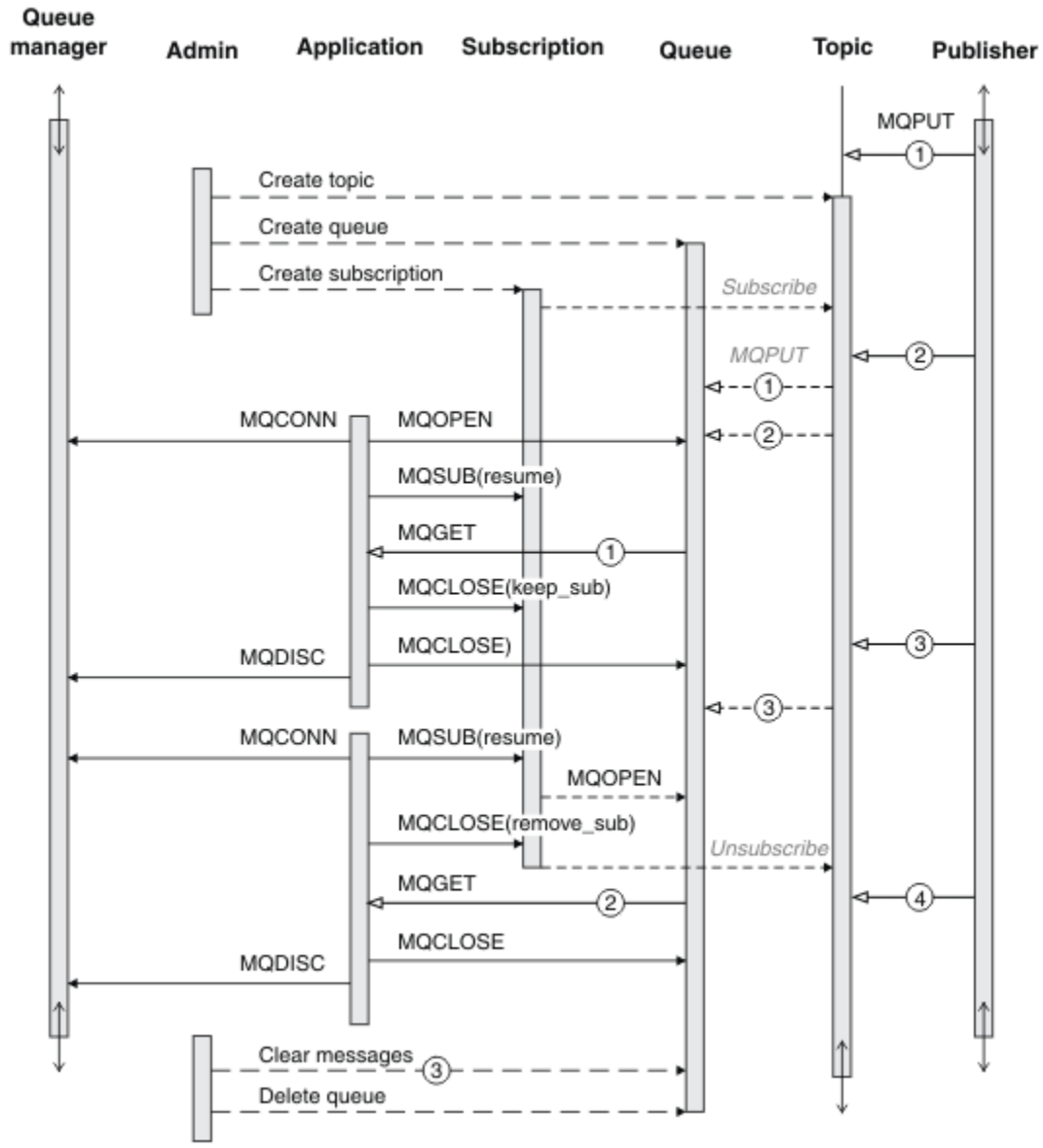


Figura 58. Líneas de vida de un suscriptor duradero no gestionado

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 la mensajería de publicación/suscripción de WebSphere 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 39. 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 39. 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 44 en la página 316 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 44. 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 44. 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.

Sin embargo, al igual que con todos los mensajes de WebSphere MQ , es posible que los mensajes, ocasionalmente, se entreguen desordenados. 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.

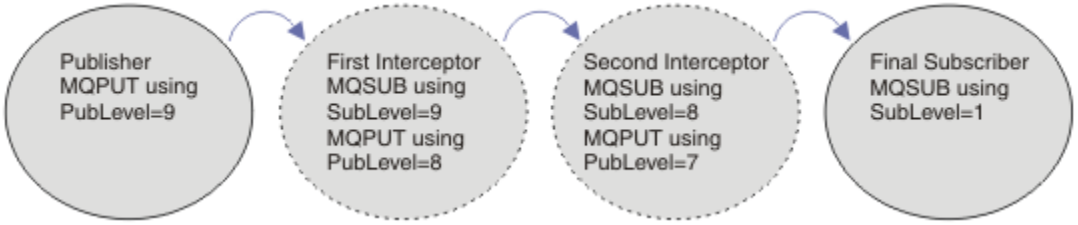


Figura 59. Secuencia de suscriptores de interceptación

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 45 en la página 318.

Tabla 45. 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 45 en la página 318 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 46 en la página 319 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 al campo **MQPMO**.`Opciones` , para combinar las opciones de colocación de mensaje.

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 319.

Tabla 46. Valores de MQPUT para mensajes republicados

Volver a publicar mensaje utilizando MQPUT	Información del mensaje de publicación
MQOD.`ObjectString`	Propiedad del mensaje `MQTopicString`
MQPMO.`Options`	Propiedad del 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 46 en la página 319. 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 60 en la página 320 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 60. Directivas de preprocesador

La Figura 61 en la página 320 lista las declaraciones que se utilizan en los fragmentos de código. Excepto para los términos resaltados, las declaraciones son estándar para una aplicación WebSphere 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 61. Declaraciones

En la Figura 62 en la página 321 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 mayoría de las estructuras de WebSphere 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.

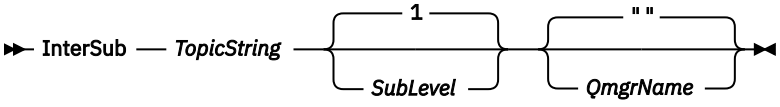
MQPRI_PRIORITY_AS_PUBLISHED
La publicación se coloca en la cola de publicación con la misma prioridad de mensaje con la que se ha publicado.

```
  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 62. Inicializaciones

La Figura 63 en la página 322 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 63. Preparación de la interceptación de publicaciones

El fragmento de código principal, Figura 64 en la página 323, 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 las propiedades **MQPMO**.opción originales 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 64. Interceptar una publicación y volver a publicarla

El fragmento de código final se muestra en la Figura 65 en la página 323.

```
    } 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 65. 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 66 en la página 324 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.

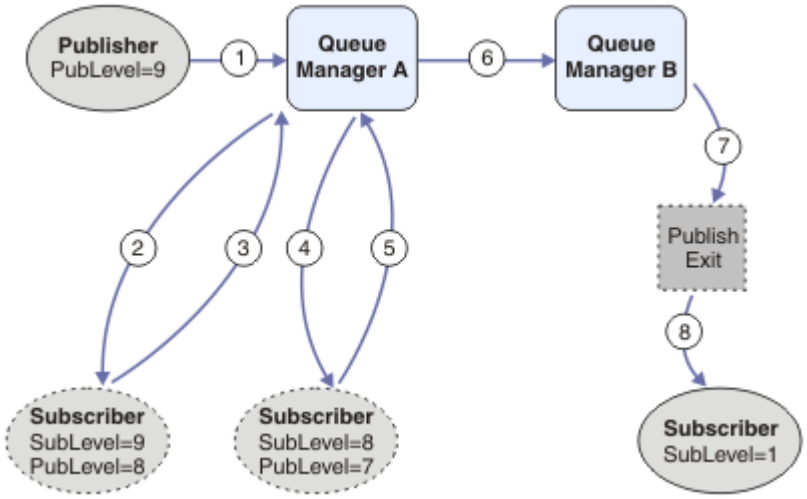


Figura 66. Salida de interceptación y publicación en un clúster

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 67 en la página 325 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 67 en la página 325, el resultado es que la publicación no es interceptada por el suscriptor utilizando `Sublevel` 9 en el gestor de colas B.

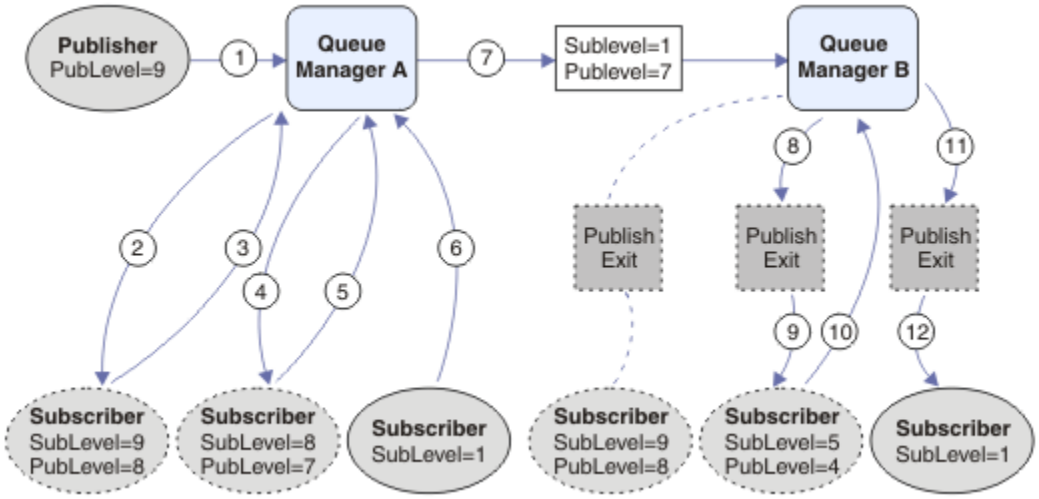


Figura 67. Despliegue complejo de suscriptores de interceptación

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 ../com.ibm.mq.ref.dev.doc/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 WebSphere MQ .

Afectan a la forma en que un gestor de colas procesa un objeto. Los atributos de cada tipo de objeto WebSphere MQ se describen en detalle en Atributos de objetos.

Algunos atributos se establecen cuando se define el objeto y solo se pueden cambiar utilizando los mandatos WebSphere MQ ; un ejemplo de este atributo es la prioridad predeterminada para 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:

Para plataformas WebSphere MQ para Windows, UNIX y Linux

El recurso MQSC, descrito en Referencia 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 utilice mandatos de WebSphere MQ para definir, modificar o visualizar los atributos, debe identificar los atributos utilizando las palabras clave que se muestran en las descripciones de los mandatos en los enlaces de tema.

Las llamadas MQINQ y MQSET utilizan matrices de selectores para identificarlos 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:

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 de una cola) o un valor constante (por ejemplo, *SyncPoint*, si el gestor de colas da soporte a puntos de sincronismo).

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 211 y "Apertura y cierre de objetos" en la página 219.

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 328
- "Algunos casos en los que falla la llamada MQINQ" en la página 329
- "Cómo establecer los atributos de cola" en la página 330

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

"Colocación de mensajes en una cola" en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

"Confirmación y restitución de unidades de trabajo" en la página 330
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 WebSphere MQ utilizando desencadenantes" en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

Consulta de los atributos de un objeto

Utilice la llamada MQINQ para consultar sobre los atributos de cualquier tipo de IBM WebSphere 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 almacenamiento intermedio la proporciona 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 son los de una cola arbitraria de clúster posible si se ha utilizado MQOO_BIND_NOT_FIXED o MQOO_BIND_ON_GROUP o se ha utilizado MQOO_BIND_AS_Q_DEF y el atributo de cola *DefBind* era MQBND_BIND_NOT_FIXED. Consulte “La llamada MQOPEN y los clústeres” en la página 356 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 devolverán los atributos de la cola alias (el objeto WebSphere 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 puede utilizar la MQI para establecer los atributos de objetos de WebSphere 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:

- Comprometer
- 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 WebSphere MQ” en la página 332.

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 WebSphere 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 WebSphere 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 se puede controlar mediante el atributo MAXUMSGS del mandato ALTER QMGR en otras plataformas, excepto z/OS. Consulte la publicación Referencia de MQSC *WebSphere MQ Consulta de mandatos de script (MQSC)* para obtener detalles de estos mandatos.

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 aquel en el que las actualizaciones que un programa ha realizado en las colas de IBM WebSphere MQ se pueden coordinar con actualizaciones 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 manejar las unidades de trabajo, IBM WebSphere 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 finalice de forma anómala, el valor de *BackoutCount* finalmente supera el 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 WebSphere MQ” en la página 332
- “Puntos de sincronismo en IBM WebSphere MQ en sistemas UNIX, Linux, and Windows” en la página 333

Conceptos relacionados
“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

“Colocación de mensajes en una cola” en la página 230

Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

"Inicio de aplicaciones de IBM WebSphere MQ utilizando desencadenantes" en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

"Cómo trabajar con clústeres y MQI" en la página 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

Consideraciones acerca del punto de sincronismo en aplicaciones IBM WebSphere MQ

Utilice esta información para obtener información acerca del uso de puntos de sincronismo en aplicaciones IBM WebSphere MQ.

La confirmación en dos fases está soportada en:

- WebSphere MQ para AIX
- WebSphere MQ para HP-UX
- WebSphere MQ para Linux
- WebSphere MQ para Solaris
- WebSphere MQ para Windows
- CICS para MVS/ESA 4.1
- CICS Transaction Server para z/OS
- TXSeries
- IMS/ESA
- Otros coordinadores externos utilizando la interfaz X/Open XA

La confirmación de una sola fase está soportada en:

- WebSphere MQ en sistemas UNIX
- WebSphere MQ para Windows

Nota: Para obtener más detalles sobre las interfaces externas, consulte "interfaces para gestores de puntos de sincronismo externos en Multiplatforms" en la página 335y la documentación de XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicada por The Open Group. Los gestores de transacciones (como por ejemplo CICS, IMS, Encinay Tuxedo) pueden participar en una confirmación en dos fases, coordinada con otros recursos recuperables. Esto significa que las funciones de cola proporcionadas por WebSphere MQ se pueden incluir en el ámbito de una unidad de trabajo, gestionada por el gestor de transacciones.

Los ejemplos que se proporcionan con WebSphere MQ muestran WebSphere MQ coordinando bases de datos compatibles con XA. Para obtener más información sobre estos ejemplos, consulte "Programas de ejemplo para plataformas distribuidas" en la página 98.

En la aplicación WebSphere MQ , puede especificar en cada llamada put y get si desea que la llamada esté bajo control de punto de sincronismo. Para hacer que una operación de transferencia opere bajo control de punto de sincronismo, utilice el valor MQPMO_SYNCPOINT en el campo *Options* de la estructura MQPMO cuando llame a MQPUT. Para una operación get, utilice el valor MQGMO_SYNCPOINT en el campo *Options* de la estructura MQGMO. Si no elige explícitamente una opción, la acción predeterminada depende de la plataforma. El valor predeterminado de control de punto de sincronismo es no.

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 *BackoutCount* alto indica un mensaje que se ha restituido repetidamente. Esto puede indicar que existe un problema con este mensaje que deberá examinar. Consulte BackoutCount para obtener detalles de *BackoutCount*.

Si un programa emite la llamada MQDISC mientras hay solicitudes no confirmadas, se produce un punto de sincronización implícito. Si el programa finaliza de forma anómala, se realiza una restitución de forma implícita.

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.

Puntos de sincronismo en IBM WebSphere MQ en sistemas UNIX, Linux, and Windows

El soporte de puntos de sincronización opera en dos tipos de unidades de trabajo: local y global.

Una unidad de trabajo *local* es aquella en la que los únicos recursos que se actualizan son los del gestor de colas de WebSphere 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. WebSphere MQ puede coordinar estas unidades de trabajo por sí mismo. También se pueden coordinar mediante un controlador de compromiso externo como, por ejemplo, otro gestor de transacciones o el controlador de compromiso de IBM i .

Para asegurar la integridad total de los datos, utilice un procedimiento de confirmación en dos fases. La confirmación en dos fases la pueden proporcionar los gestores de transacciones compatibles con XA y bases de datos como TXSeries y UDB. Los productos WebSphere MQ (excepto WebSphere MQ para IBM i y WebSphere MQ para z/OS) pueden coordinar unidades de trabajo globales utilizando un proceso de confirmación de dos fases.

Unidades de trabajo locales

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 WebSphere 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 de trabajo globales

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 WebSphere MQ para IBM i o WebSphere MQ para z/OS. No está soportado en un entorno de cliente MQI de WebSphere MQ.

Aquí, WebSphere 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 de contexto representa la conexión con el gestor de colas de WebSphere 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 de dos fases en el que los gestores de recursos (por ejemplo, gestores de bases de datos compatibles con XA como DB2, Oracley Sybase) deben prepararse para la confirmación. 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 sincronismo distinto de WebSphere MQ ; por ejemplo, CICS, Encinao Tuxedo.

En esta situación, WebSphere MQ en sistemas UNIX and Linux y WebSphere MQ para 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 de obtención o colocación 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 las llamadas de API de WebSphere MQ en programas CICS , tanto si elige utilizar las capacidades XA de CICS. Si no utiliza XA, las transferencias y obtenciones de mensajes hacia y desde las colas no se gestionarán dentro de las unidades atómicas de trabajo 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 utiliza XA, CICS utiliza un protocolo de confirmación de 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” en la página 42y también la documentación de TXSeries CICS , por ejemplo, *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

interfaces para gestores de puntos de sincronismo externos en Multiplatforms

WebSphere MQ en sistemas UNIX and Linux , y WebSphere MQ para Windows dan soporte a la coordinación de transacciones por parte de gestores de puntos de sincronización 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. El gestor de recursos para WebSphere MQ en sistemas UNIX, Linux y Windows se denomina MQSeries_XA_RMI. Para obtener más detalles sobre las interfaces XA, consulte la documentación de XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publicada por The Open Group.

En una configuración XA, los sistemas WebSphere MQ en UNIX, Linuxy Windows cumplen el rol de un Resource ManagerXA. 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 Resource Manager 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:

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 estructura de conmutación de IBM WebSphere MQ XA" en la página 71.

El método que debe utilizarse para enlazarlos a un coordinador de punto de sincronismo XA lo define el coordinador; consulte la documentación proporcionada por dicho coordinador para determinar cómo habilitar WebSphere MQ para cooperar con el coordinador de punto 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 en WebSphere MQ el nombre del gestor de transacciones, 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 217.
2. En sistemas Windows , todas las funciones declaradas en el conmutador XA se declaran como funciones _cdecl.
3. 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 cliente JMS (aplicación CLIENT JEE) que se ejecuta en un servidor JEE no tiene esta restricción, por lo que una única transacción gestionada por el servidor JEE puede coordinar varios gestores de colas en la misma transacción. Sin embargo, una aplicación de servidor JMS, que se ejecuta en modalidad de enlaces, sigue sujeta a la regla de que sólo se permite una conexión a la vez.
4. 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, pueden utilizar la salida UE014015 para TXSeries CICS.

Inicio de aplicaciones de IBM WebSphere MQ utilizando desencadenantes

Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

Algunas aplicaciones de WebSphere MQ que dan servicio a las colas se ejecutan continuamente, por lo que siempre están disponibles para recuperar los 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.

WebSphere MQ proporciona un recurso que permite que una aplicación se inicie automáticamente cuando hay mensajes disponibles para recuperar. 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 a una cola se proporciona mediante el atributo de 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.

El desencadenamiento está soportado por clientes WebSphere MQ que se ejecutan en los entornos siguientes:

- Sistemas UNIX and Linux
- Sistemas Windows

Una aplicación que se ejecuta en un entorno de cliente es la misma que la que se ejecuta en un entorno de WebSphere 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 sistemas AIX, HP-UX, IBM i, Solaris, z/OSo Windows , 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 se puede producir en otras ocasiones (consulte “Condiciones para un suceso desencadenante” en la página 343). WebSphere MQ tiene un rango de opciones que le permiten controlar las condiciones que provocan un suceso desencadenante (consulte “Control de sucesos desencadenantes” en la página 347).

Mensaje de desencadenante
El gestor de colas crea un *mensaje desencadenante* cuando reconoce un suceso desencadenante (consulte “Condiciones para un suceso desencadenante” en la página 343). 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 354).

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 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. Una cola compartida, una cola local accesible por los gestores de colas de un grupo de compartición de colas, puede ser una cola de inicio en WebSphere MQ para 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. En z/OS, el iniciador de canal normalmente se inicia manualmente, o se puede realizar automáticamente cuando se inicia un gestor de colas cambiando CSQINP2 en el JCL de inicio del gestor de colas. En otras plataformas, se inicia automáticamente cuando se inicia el gestor de colas o se puede iniciar manualmente con el mandato runmqchi.

(Para obtener más información, consulte “Proceso de cola de inicio por parte de supervisores desencadenantes” en la página 351.)

Para entender cómo funciona el mecanismo de desencadenamiento, consulte la Figura 68 en la página 339, que es un ejemplo de tipo desencadenante FIRST (MQTT_FIRST).

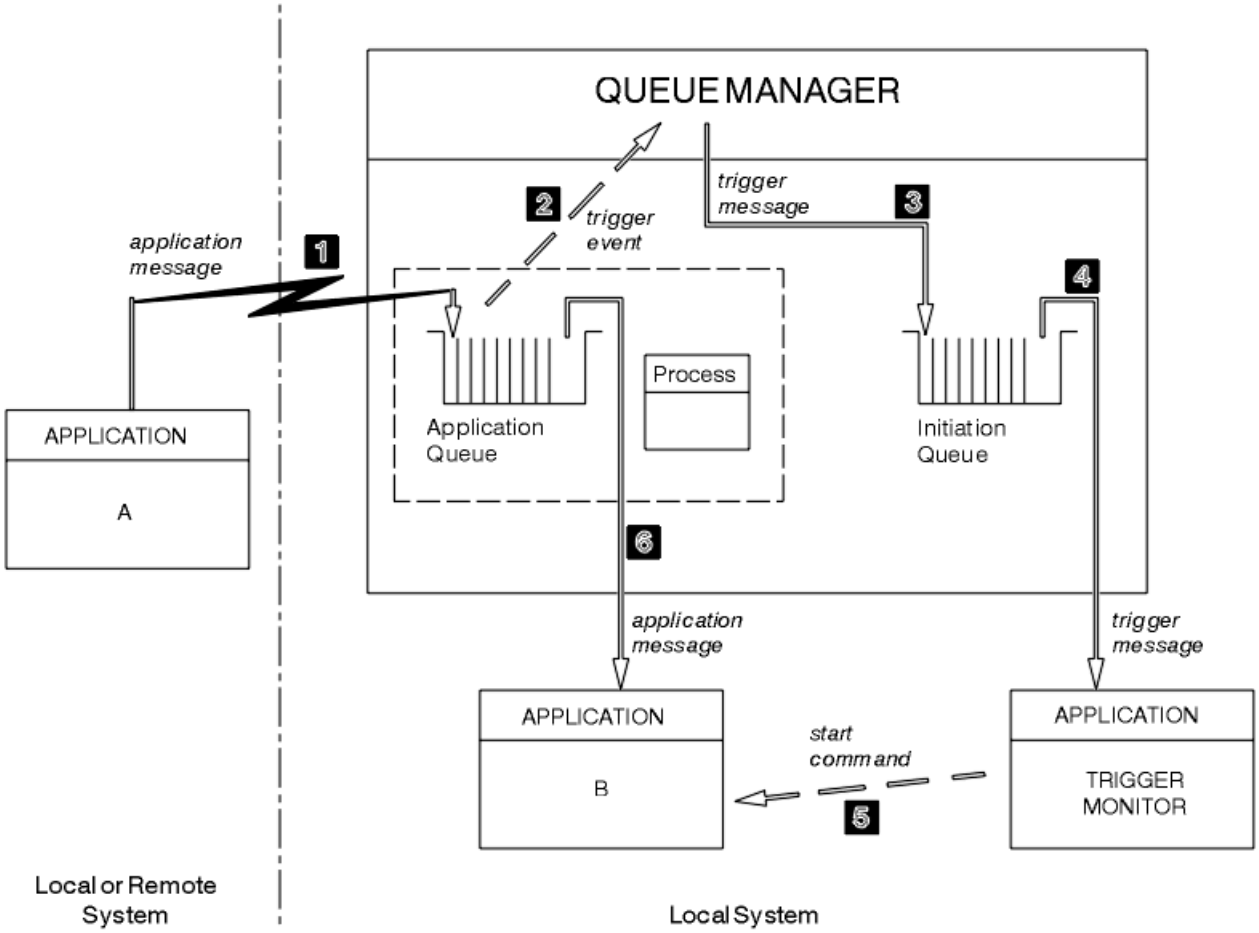


Figura 68. Flujo de aplicaciones y mensajes desencadenantes

En la Figura 68 en la página 339, 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 69 en la página 340.

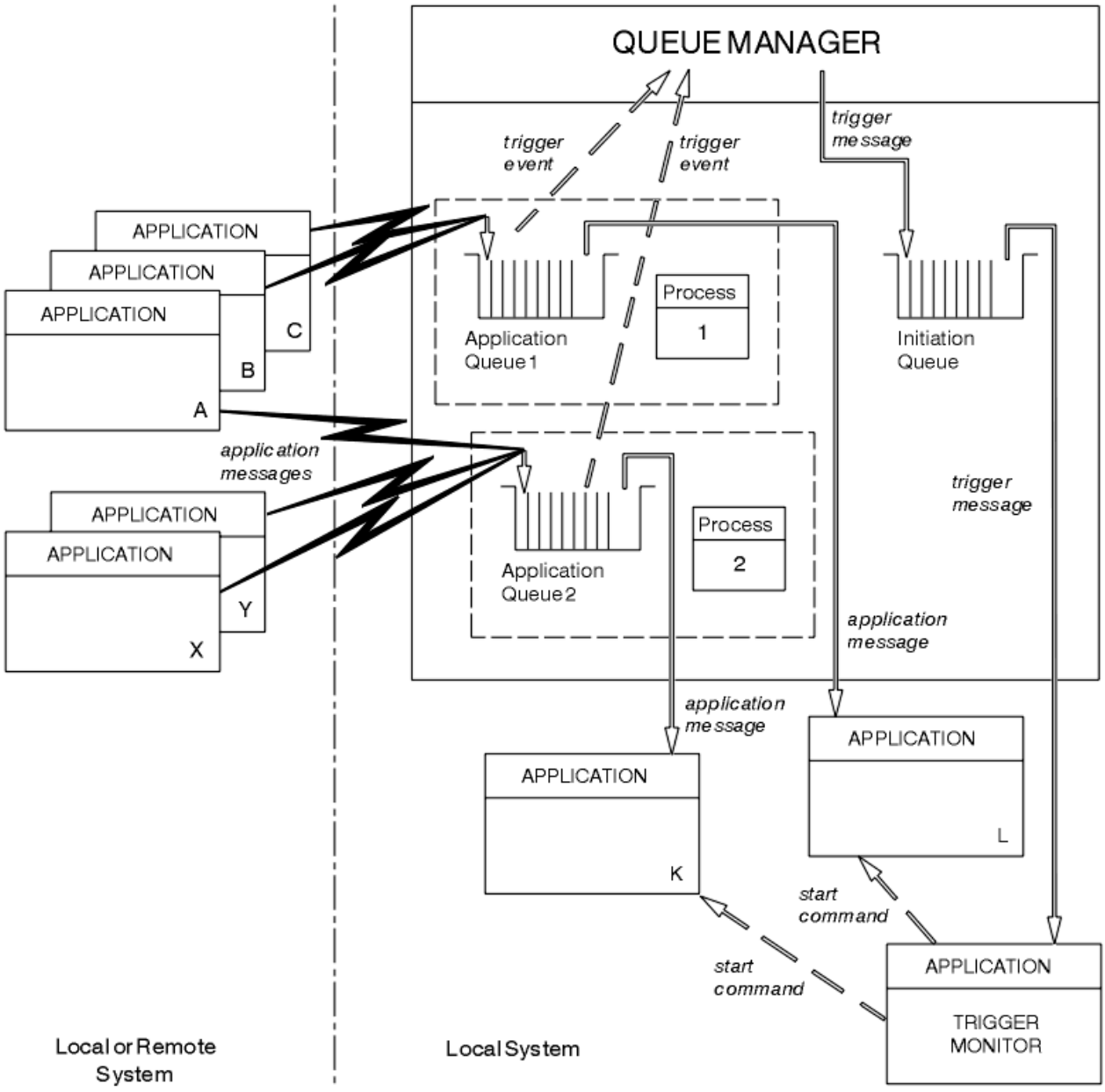


Figura 69. Relación de colas dentro del desencadenante

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 cómo iniciar aplicaciones WebSphere MQ utilizando desencadenantes:

- "Requisitos previos del desencadenamiento" en la página 341
- "Condiciones para un suceso desencadenante" en la página 343
- "Control de sucesos desencadenantes" en la página 347
- "Diseño de una aplicación que utiliza las colas desencadenadas" en la página 349
- "Proceso de cola de inicio por parte de supervisores desencadenantes" en la página 351
- "Propiedades de los mensajes desencadenantes" en la página 353
- "Cuando el desencadenamiento no funciona" en la página 355

Conceptos relacionados

"Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)" en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

"Conexión y desconexión de un gestor de colas" en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

"Colocación de mensajes en una cola" en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

"Obtención de mensajes de una cola" en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

"Consulta y establecimiento de atributos de objeto" en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

"Confirmación y restitución de unidades de trabajo" en la página 330
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 355
Hay opciones especiales en las llamadas y códigos de retorno relacionados con las agrupaciones en clúster.

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. Realice una de las siguientes acciones:
   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 existe y que la aplicación puede utilizar (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)
```

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')
```

   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
Sistemas UNIX, Linuxy Windows	Utilice DEFINE PROCESS o utilice SYSTEM.DEFAULT.PROCESS y modifíquelo utilizando ALTER PROCESS

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 se puede dejar en blanco. Excepto en IBM WebSphere 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 desencadenante.

5. Si ha creado un objeto de definición de proceso para especificar propiedades de la aplicación que va a servir a la cola de aplicación, asocie el objeto de proceso con la cola de aplicación nombrándolo en el atributo *ProcessName* de la cola.

Plataforma	Utilizar los mandatos
Sistemas UNIX, Linuxy Windows	ALTER QLOCAL

6. Las instancias de inicio los supervisores desencadenantes 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 351.

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 más información, consulte “Control de sucesos desencadenantes” en la página 347.

Condiciones para un suceso desencadenante

Las referencias a colas compartidas en este tema significan colas compartidas en un grupo de compartición de colas, sólo disponibles en WebSphere MQ para z/OS.

El gestor de colas crea un mensaje desencadenante cuando se cumplen las condiciones siguientes:

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 establece en el atributo de cola local *TriggerMsgPriority* ; si se establece en cero, cualquier mensaje se califica.
3. El número de mensajes en la cola con prioridad mayor o igual que *TriggerMsgPriority* era anteriormente, en función 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 de espera con un *WaitInterval*adecuado, para 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. Para el desencadenamiento de tipo FIRST o DEPTH, ningún programa tiene la cola de aplicación abierta para eliminar 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 para una cola debe ser cero para que ese gestor de colas genere un mensaje desencadenante para la cola. No obstante, si algún gestor de colas del grupo de compartición de colas tiene la cola abierta al haber utilizado la opción MQOO_INPUT_EXCLUSIVE, ningún gestor de colas del grupo de compartición generará ningún mensaje desencadenante para esa cola.

      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 WebSphere MQ para z/OS, si la cola de aplicación es una con un atributo *Usage* de MQUS_NORMAL, las solicitudes get para la misma no se inhiben (es decir, el atributo de cola *InhibitGet* es MQQA_GET_ALLOWED). Además, si la cola de aplicación desencadenada es una con un atributo *Usage* de MQUS_XMITQ, las solicitudes get para la misma no se inhiben.
6. Realice una de las siguientes acciones:
   - El atributo de cola local *ProcessName* para 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* para la cola está en blanco, pero la cola es una cola de transmisión. Puesto que la definición de proceso es opcional, el atributo *TriggerData* también puede contener el nombre del canal que se va a iniciar. En este caso, el mensaje desencadenante contiene atributos con los valores siguientes:
     - *QName*: nombre de cola
     - *ProcessName*: espacios en blanco
     - *TriggerData*: datos de desencadenante
     - *ApplType*: MQAT_UNKNOWN
     - *ApplId*: espacios en blanco
     - *EnvData*: espacios en blanco
     - *UserData*: espacios en blanco
7. Se ha creado una cola de inicio y se ha especificado en el atributo de cola local *InitiationQName* . Además:
   - Las solicitudes de obtención no están inhibidas para la cola de inicio (es decir, el atributo de cola *InhibitGet* es MQQA_GET_ALLOWED).
   - Las solicitudes de colocación no deben estar inhibidas para la cola de inicio (es decir, el atributo de cola *InhibitPut* debe ser MQQA_PUT_ALLOWED).
   - El atributo *Usage* de la cola de inicio debe 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. Un supervisor desencadenante tiene actualmente la cola de inicio abierta para eliminar mensajes (es decir, el atributo de cola local *OpenInputCount* es mayor que cero).
9. El control desencadenante (atributo de cola local*TriggerControl*) para la cola de aplicación se establece en MQTC_ON. Para ello, establezca el atributo *trigger* cuando defina la cola o utilice el mandato 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. Se coloca un mensaje adecuado en la cola, para un *TriggerType* de 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 343 a “10” en la página 344 (excluyendo “3” en la página 343), si en el caso de MQTT_FIRST ha transcurrido un intervalo suficiente (atributo de gestor de colas*TriggerInterval*) desde que se grabó el último mensaje desencadenante para 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 detiene y reinicia el gestor de colas, se restablece el *TriggerInterval temporizador* . Hay una pequeña ventana durante la cual es posible generar dos mensajes de desencadenante. La ventana existe cuando el atributo desencadenante de la cola se establece en habilitado al mismo tiempo que llega un mensaje y la cola no estaba vacía anteriormente (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* de MQTT_FIRST o MQTT_DEPTH, y hay al menos:

    - Un (MQTT_FIRST), o
    - *TriggerDepth* (MQTT_DEPTH)

    mensajes en la cola de prioridad suficiente (condición “2” en la página 343) y las condiciones de “6” en la página 344 a “10” en la página 344 también se cumplen.

    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 produzca un bucle de este tipo, utilice el campo *BackoutCount* de MQMD para detectar mensajes que se restituyen repetidamente. Para más información, consulte “Mensajes que se restituyen” en la página 38.

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 el valor de *TriggerType*, *TriggerMsgPriority*o *TriggerDepth* (si es relevante) ha cambiado,

       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 343) y las condiciones “4” en la página 343 a “10” en la página 344 (excluyendo “8” en la página 344) también se cumplen.

       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 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 343) en cualquiera de las colas de las que esta es la cola de inicio y también se cumplen las condiciones “4” en la página 343 a “10” en la página 344. (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 atributo de cola *InhibitGet* de una cola de aplicación cambia de MQQA_GET_inhibiITED 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 343) en la cola y también se cumplen las condiciones “4” en la página 343 a “10” en la página 344, excluyendo “5” en la página 344.

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 343) 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 343 a “10” en la página 344 (excluyendo “8” en la página 344), 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 *TriggerMsgPriority* se establece en un valor mayor que la prioridad predeterminada de la cola, no se desencadenará ningún mensaje. Si *TriggerMsgPriority* se establece en un valor igual o menor que la prioridad predeterminada de la cola, el desencadenamiento se produce para el tipo FIRST, EVERY y DEPTH. Para obtener información sobre estos tipos, consulte la descripción del campo *TriggerType* en “Control de sucesos desencadenantes” en la página 347.

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 coloca 100 mensajes de prioridad inferior a *TriggerMsgPriority*, la profundidad de cola efectiva para fines de desencadenamiento sigue siendo cero. Si, a continuación, coloca otro mensaje en la cola, pero esta vez la prioridad es mayor o igual que *TriggerMsgPriority*, la profundidad de cola efectiva aumenta de cero a uno y se cumple la condición para *TriggerType* FIRST.

Nota:

1. En el paso “12” en la página 345 (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, sólo se genera un mensaje desencadenante.
2. Si WebSphere MQ segmenta un mensaje durante MQPUT, no se procesará un suceso desencadenante hasta que todos los segmentos se hayan colocado correctamente en la cola. Sin embargo, una vez que los segmentos de mensajes están en la cola, WebSphere MQ los trata como mensajes individuales

para fines de desencadenamiento. 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 hace que se procesen sus propios sucesos desencadenantes a medida que se mueven a través de la red de WebSphere MQ .

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 se 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 igual que establecer *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.
PRIMERO	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 349.

PROFUNDIDAD	Un suceso desencadenante sólo se produce 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 < xph> < pv>TriggerControl< /pv> < /xph>) 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 343.

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 de gestor de colas *TriggerInterval*. El valor predeterminado es 999 999 999 milisegundos.
- En WebSphere MQ para z/OS, las colas de aplicación que dan nombre a una cola de inicio abierta se exploran periódicamente. Si han pasado *TRIGINT* milisegundos desde que se envió el último mensaje desencadenante y la cola cumple 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 sirva a la cola de aplicaciones, el tipo de desencadenante FIRST podría 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 WebSphere MQ para z/OS si establece *TRIGINT* en un valor bajo, y no hay ninguna aplicación que sirva al tipo de desencadenante FIRST cola de aplicación, el desencadenamiento de backstop generará un mensaje desencadenante cada vez que tenga lugar la exploración periódica de las colas de aplicación que denominan colas de inicio abiertas.
- Si se restituye una unidad de trabajo (consulte Desencadenar mensajes y unidades de trabajo) y el intervalo de desencadenante se ha establecido en un valor alto (o el valor predeterminado), se genera un mensaje desencadenante cuando se restituye 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.

   Sin embargo, esto significa que a veces se crean mensajes desencadenantes cuando las condiciones para un suceso desencadenante *no* se satisfacen. 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* en un valor adecuado.

   Los mensajes desencadenantes creados siempre están disponibles, tanto si se restituye como si se confirma la unidad de trabajo.
2. En el caso de las colas locales compartidas (es decir, colas compartidas en un grupo de compartición de colas) el gestor de colas solo cuenta los 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. *WaitInterval* debe ser suficiente para permitir el tiempo razonable más largo entre la colocación de un mensaje y la confirmación de la 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.

Supervisores desencadenantes en sistemas UNIX y Windows

Este tema contiene información sobre los supervisores desencadenantes que se proporcionan en los sistemas UNIX y 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 “Programas de ejemplo para plataformas distribuidas” en la página 98 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. El *ApplId* de la definición PROCESS relevante (si se ha creado)
  2. La estructura MQTMC2, delimitada por comillas dobles
  3. El *EnvData* de la definición PROCESS relevante (si se ha creado)

  donde *ApplId* es el nombre del programa que se debe ejecutar tal como se especificaría 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 WebSphere MQ sobre el número de aplicaciones que se pueden ejecutar.

Para ejecutar la aplicación iniciada en segundo plano en sistemas Windows , en el campo *ApplId* , prefije el nombre de la aplicación con un mandato START. Por ejemplo:

```
START ?B AMQSECHA
```

Para ejecutar la aplicación iniciada en segundo plano en sistemas UNIX , ponga un & al final del *EnvData* de la definición PROCESS.

Nota: Cuando una vía de acceso de Windows tiene espacios como parte del nombre de vía de acceso, éstos deben estar entre comillas (") para asegurarse de que se maneja 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 entre comillas dobles. START especifica que el primer argumento entre comillas dobles se tratará como el título del nuevo mandato. Para asegurarse de que Windows no confunde la vía de acceso de la aplicación con un argumento 'title', añada una serie de título entre comillas dobles al mandato antes del nombre de la aplicación.

Se proporcionan los siguientes supervisores desencadenantes para el cliente WebSphere MQ :

runmqtmc
Es lo mismo que runmqtrm excepto que enlaza con las bibliotecas de cliente MQI de WebSphere MQ .

Para CICS
El supervisor desencadenante amqltmc0 se proporciona para CICS. Funciona de la misma forma que el supervisor desencadenante estándar, runmqtrm, pero lo ejecuta de una forma diferente y desencadena transacciones CICS .

Este tema sólo se aplica a los sistemas Windows, UNIXy Linux .

Se proporciona como un programa CICS ; defínalo 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 WebSphere MQ para 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, porque 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 se utiliza la estructura MQTMC2 , sólo es necesario proporcionar los parámetros *StrucId*, *Version*, *QName*y *QMgrName* al 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 es MQAT_CICS. La lectura de mensajes de la cola de inicio se realiza bajo el control de punto de sincronización 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.

A continuación se indican las versiones disponibles del supervisor desencadenante:

Versión	**Uso**
amqltmc0	TXSeries para AIX, HP-UXy Sun Solaris Versión 5.1
amqltmc4	TXSeries para Windows, Versión 5.1
amqltmcc	Versión enlazada 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.
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. Para ello, debe denominar el objeto de definición de proceso (si se ha creado) en el atributo *ProcessName* de la cola.

Para obtener más información sobre la interfaz de supervisor desencadenante, consulte MQTMC2.

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 353
- “Reinicio del gestor de colas y mensajes desencadenantes” en la página 354
- “Mensajes desencadenantes y atributos de objetos” en la página 354
- “Formato de los mensajes desencadenantes” en la página 354

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 WebSphere MQ excepto WebSphere MQ for z/OS, este campo se puede utilizar para especificar el nombre del canal que se va a 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 o CSQQTRMN proporcionado por WebSphere MQ para z/OS, el atributo *ApplId* del objeto de definición de proceso es un identificador de transacción CICS o 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*. Los supervisores desencadenantes proporcionados por WebSphere MQ for z/OS(CKTI o CSQQTRMN) no utilizan este campo, pero otros supervisores desencadenantes pueden 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 correctamente, el mensaje desencadenante se descarta 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 un programa de utilidad de rastreo (por ejemplo, CICS AUXTRACE si el programa se ejecuta bajo CICS) para encontrar la causa de la anomalía.

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 356
- “MQPUT, MQPUT1 y clústeres” en la página 357
- “MQINQ y clústeres” en la página 357
- “MQSET y clústeres” en la página 358
- “Códigos de retorno” en la página 358

Conceptos relacionados

“Descripción general de la interfaz de cola de mensajes (Message Queue Interface, MQI)” en la página 199
Conozca los componentes de la interfaz de colas de mensajes (MQI).

“Conexión y desconexión de un gestor de colas” en la página 211
Para utilizar los servicios de programación de WebSphere 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 219
Esta información proporciona información sobre cómo abrir y cerrar objetos de WebSphere MQ .

“Colocación de mensajes en una cola” en la página 230
Utilice esta información para conocer cómo poner mensajes en una cola.

“Obtención de mensajes de una cola” en la página 245
Utilice esta información para aprender a obtener mensajes de una cola.

“Consulta y establecimiento de atributos de objeto” en la página 327
Los atributos son las propiedades que definen las características de un objeto WebSphere MQ .

“Confirmación y restitución de unidades de trabajo” en la página 330
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 WebSphere MQ utilizando desencadenantes” en la página 337
Información sobre los desencadenantes y cómo iniciar las aplicaciones de IBM WebSphere MQ utilizando desencadenantes.

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 Equilibrio de carga de trabajo.

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 WebSphere 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, 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 MQOPEN opciones MQOO_BROWSE, MQOO_INPUT_*o MQOO_SET requieren una instancia local de la cola de clúster para que MQOPEN sea satisfactorio.

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 225

El enlace utilizado para el descriptor de contexto de cola se toma del atributo de cola *DefBind* , que puede tomar el valor MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDo 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 de forma inesperada o no responde a tiempo.

Se graba un mensaje en el registro del sistema en WebSphere MQ for z/OS 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, la salida de carga de trabajo del clúster no se puede cargar.

Se escribe un mensaje en el registro del sistema y el proceso continúa como si el atributo ClusterWorkloadExit estuviera en blanco.

En plataformas distintas de z/OS, se emite una llamada MQCONN o MQCONNX para conectarse a un gestor de colas. La llamada falla porque la salida de carga de trabajo de clúster, definida por el atributo ClusterWorkloadExit del gestor de colas, no se puede cargar.

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 sólo se produce cuando se cumplen las dos condiciones siguientes:

- 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.

Escritura de aplicaciones cliente

Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

Las aplicaciones se pueden crear y ejecutar en el entorno de cliente WebSphere MQ . La aplicación debe estar compilada y enlazada con el cliente MQI de WebSphere MQ utilizado. 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 clientes MQI de WebSphere MQ” en la página 366.

Puede ejecutar una aplicación WebSphere MQ en un entorno completo de WebSphere MQ y en un entorno de cliente MQI de WebSphere MQ sin cambiar el código, siempre que se cumplan determinadas condiciones. Para obtener más información sobre cómo ejecutar las aplicaciones en el entorno de cliente WebSphere MQ , consulte “Ejecución de aplicaciones en el entorno de cliente MQI de IBM WebSphere MQ” en la página 368.

Si utiliza la interfaz de cola de mensajes (MQI) para escribir aplicaciones para que se ejecuten en un entorno de cliente MQI de WebSphere MQ , existen algunos controles adicionales que se deben imponer durante una llamada MQI para asegurarse de que el proceso de la aplicación WebSphere MQ no se interrumpe. Para obtener más información sobre estos controles, consulte “Utilización de la interfaz de cola de mensajes (MQI) en una aplicación cliente” en la página 361.

Consulte los temas siguientes para obtener información sobre la preparación y ejecución de otros tipos de aplicaciones como aplicaciones cliente:

- “Preparación y ejecución de aplicaciones CICS y Tuxedo” en la página 379
- “Preparación y ejecución de aplicaciones de Microsoft Transaction Server” en la página 41
- “Preparación y ejecución de aplicaciones JMS de WebSphere MQ” en la página 382

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 8
Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

“Decidir qué lenguaje de programación utilizar” en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

“Diseño de aplicaciones IBM WebSphere MQ” en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

“Programas WebSphere MQ de ejemplo” en la página 98
Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

“Escritura de una aplicación de gestión de colas” en la página 199
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.

“Utilización de servicios web en WebSphere MQ” en la página 965
Puede desarrollar aplicaciones IBM WebSphere MQ para servicios web utilizando el transporte IBM WebSphere MQ para SOAP o el puente IBM WebSphere MQ para HTTP.

“Escritura de aplicaciones de publicación/suscripción” en la página 284
Empiece a escribir aplicaciones de publicación/suscripción de WebSphere MQ .

“Creación de una aplicación IBM WebSphere MQ” en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

“Manejo de errores de programa” en la página 559
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.

Utilización de la interfaz de cola de mensajes (MQI) en una aplicación cliente

Esta colección de temas tiene en cuenta las diferencias entre escribir la aplicación WebSphere MQ para que se ejecute en un entorno de cliente MQI de WebSphere MQ y para que se ejecute en el entorno de gestor de colas de WebSphere MQ completo.

Cuando diseñe una aplicación, tenga en cuenta qué controles debe imponer durante una llamada MQI para asegurarse de que el proceso de la aplicación WebSphere MQ no se interrumpe.

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.

En plataformas distintas de z/OS, puede aumentar el atributo de longitud máxima de mensaje de un gestor de colas. Los detalles se proporcionan en 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 WebSphere MQ intenta utilizar la llamada MQPUT o la llamada MQGET con un mensaje mayor que este, 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.

Elección del identificador de juego de caracteres codificado (CCSID) de cliente o servidor

Utilice el CCSID local para el cliente. El gestor de colas realiza la conversión necesaria. Utilice 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 pasados a través de la MQI desde la aplicación al apéndice de cliente deben estar en el CCSID local, codificado para el cliente MQI de WebSphere MQ . 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.

Sin embargo, el cliente Java en V7puede realizar la conversión si el gestor de colas no puede hacerlo. Consulte “ WebSphere MQ Clases para conexiones de cliente Java” en la página 682

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 requerido, se puede alterar temporalmente con la variable de entorno MQCCSID, utilizando uno de estos mandatos:

- En Windows:

```
SET MQCCSID=850
```

- En sistemas UNIX:

```
export MQCCSID=850
```

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 PUT que incluyen cabeceras WebSphere MQ después del descriptor de mensaje (MQMD), tenga en cuenta que el CCSID y los campos de codificación del MQMD se sobrescriben después de completar la primera PUT.

Después del primer PUT, estos campos contienen el valor utilizado por el gestor de colas conectado para convertir las cabeceras de WebSphere 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.

MaxMsgLongitud
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 WebSphere MQ, uno de los roles del gestor de colas es el control de punto de sincronización dentro de una aplicación. Si una aplicación se ejecuta en un cliente base de WebSphere MQ , puede emitir MQCMIT y MQBACK, pero el ámbito del control de punto de sincronización está limitado a los recursos MQI. El verbo WebSphere MQ MQBEGIN 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 proporcionado con productos WebSphere MQ u otro supervisor de transacciones como 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 WebSphere 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 de 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 llama a MQOPEN con MQOO_READ_AHEAD, el cliente de WebSphere MQ sólo habilita la lectura anticipada si se cumplen determinadas condiciones. Estas condiciones incluyen:

- El cliente y el gestor de colas remoto deben ser de WebSphere MQ Versión 7 o posterior.
- La aplicación cliente debe compilarse y enlazarse con las bibliotecas de cliente MQI WebSphere 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 264

La configuración de lectura anticipada está controlada por tres atributos, MaximumSize, PurgeTimey UpdatePercentage, que se especifican en la stanza MessageBuffer del archivo de configuración del cliente de WebSphere 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 118.

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/SSL también se produce para 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 el programa de salida se inicializa para la segunda conversación, aunque puede alterar los campos MQCD, *no* el canal actúa sobre ellos. 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 Utilización de la estructura MQCNO en una llamada MQCONNX. 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 405.

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 descriptor de conexión compartido en un cliente MQI de WebSphere MQ para conectarse a un gestor de colas de servidor que no admita descriptores de conexión compartidos.

Para más información, consulte “Utilización de MQCONNX” en la página 365.

Creación de aplicaciones para clientes MQI de WebSphere MQ

Las aplicaciones se pueden crear y ejecutar en el entorno de cliente MQI de WebSphere MQ . La aplicación debe estar compilada y enlazada con el cliente MQI de WebSphere MQ utilizado. 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 47. Lenguajes de programación soportados en los entornos de cliente

Plataforma de cliente	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Sí	Sí	Sí			
HP Integrity NonStop Server	Sí		Sí	Sí		
HP-UX	Sí	Sí	Sí			
Linux	Sí	Sí	Sí			
Solaris	Sí	Sí	Sí			
Windows	Sí	Sí	Sí			Sí

Consulte los temas relacionados para obtener instrucciones sobre cómo enlazar o crear aplicaciones cliente en estos idiomas.

Enlace de aplicaciones C con el código de cliente MQI de WebSphere MQ

Después de haber escrito la aplicación WebSphere MQ que desea ejecutar en el cliente MQI de WebSphere MQ , debe enlazarla a un gestor de colas.

Puede enlazar la aplicación a un gestor de colas de dos maneras:

1. Directamente, en cuyo caso el gestor de colas debe estar en la misma estación de trabajo que la aplicación
2. A un archivo de biblioteca de cliente, que le proporciona acceso a gestores de colas en la misma estación de trabajo o en otra distinta

WebSphere MQ proporciona un archivo de biblioteca de cliente para cada entorno:

AIX
La biblioteca libmqic.a para aplicaciones sin hebras o la biblioteca libmqic_r.a para aplicaciones con hebras.

HP-UX
La biblioteca libmqic.sl para aplicaciones sin hebras o biblioteca libmqic_r.sl para aplicaciones con hebras.

Linux
La biblioteca libmqic.so para aplicaciones sin hebras o biblioteca libmqic_r.so para aplicaciones con hebras.

Solaris
libmqic.so.

Si desea utilizar los programas en una estación de trabajo que solo tiene instalado el cliente MQI de WebSphere MQ para Solaris, debe volver a compilar los programas para enlazarlos con la biblioteca de cliente:

```
$ /opt/SUNWspro/bin/cc -o <prog> <prog> c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Hay que especificar los parámetros en el orden correcto, tal como se indica.

Windows
MQIC32.LIB.

Enlace de aplicaciones C++ con el código de cliente MQI WebSphere MQ

Puede escribir aplicaciones para ejecutarlas en el cliente en C++. Los métodos de compilación varían según el entorno.

Para obtener información sobre cómo enlazar las aplicaciones C++, consulte Creación de programas C++ de WebSphere MQ.

Para obtener detalles completos sobre todos los aspectos del uso de C++, consulte Utilización de C++

Enlace de aplicaciones COBOL con el código de cliente MQI de IBM WebSphere MQ

Después de haber escrito una aplicación COBOL que desea ejecutar en el cliente MQI de IBM WebSphere MQ , debe enlazarla con una biblioteca adecuada.

IBM WebSphere MQ proporciona un archivo de biblioteca de cliente para cada entorno:

AIX
Enlace una aplicación COBOL sin hilos con la biblioteca libmqicb.a o una aplicación COBOL con hilos con libmqicb_r.a.

HP-UX
Enlace una aplicación COBOL sin hilos con la biblioteca libmqicb.sl o una aplicación COBOL con hilos con libmqicb_r.sl.

Linux
Enlace una aplicación COBOL sin hilos con la biblioteca libmqicb.so o una aplicación COBOL con hilos con libmqicb_r.so.

Solaris
Enlace una aplicación COBOL sin hilos con la biblioteca libmqicb.so o una aplicación COBOL con hilos con libmqicb_r.so.

Windows
Enlace el código de aplicación con la biblioteca MQICCBB para COBOL de 32 bits. El cliente MQI de IBM WebSphere MQ para Windows no da soporte a COBOL de 16 bits.

Enlace de aplicaciones de Visual Basic con el código de cliente MQI de WebSphere MQ

Puede enlazar aplicaciones Visual Basic con el código de cliente MQI de WebSphere MQ en Windows.

Enlace la aplicación Visual Basic con los siguientes archivos de inclusión:

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 asegurarse de que la selección automática correcta de la dll de cliente:

MQIC32.dll
Windows 2000, Windows XP y Windows 2003

Ejecución de aplicaciones en el entorno de cliente MQI de IBM WebSphere MQ

Puede ejecutar una aplicación IBM WebSphere MQ tanto en un entorno IBM WebSphere MQ completo como en un entorno de cliente MQI de IBM WebSphere MQ 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 del gestor de colas no tiene como prefijo un asterisco (*) en una llamada `MQCONN` o `MQCONNX` .
- La aplicación no necesita utilizar ninguna de las excepciones listadas en ¿Qué aplicaciones se ejecutan en un cliente MQI de IBM WebSphere MQ ?

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 cliente MQI de IBM WebSphere MQ , recuerde que:

- Cada aplicación que se ejecuta en el entorno de cliente MQI de IBM WebSphere MQ 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 cliente MQI de IBM WebSphere MQ a gestores de colas

Una aplicación que se ejecuta en un entorno de cliente MQI de IBM WebSphere MQ puede conectarse a un gestor de colas de varias formas. 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 WebSphere 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 WebSphere 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 proporciona). 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 432.
2. Si se establece la variable de entorno MQSERVER, se utiliza el canal que define.

3. Si se define un archivo `mqclient.ini` y contiene un ServerConnectionParms, se utiliza el canal que define. Para obtener más información, consulte Configuración de un cliente utilizando un archivo de configuración y Stanza CHANNELS del archivo de configuración de 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.
5. Si se define un archivo `mqclient.ini` y 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 Configuración de un cliente utilizando un archivo de configuración y Stanza CHANNELS del archivo de configuración de cliente.
6. Finalmente, si las variables de entorno no se establecen, el cliente busca una tabla de definiciones de canal de cliente con una vía de acceso y un nombre que estén establecidos a partir de `DefaultPrefix` en el archivo `mqs.ini`. Si la búsqueda de una tabla de definición de cliente falla, el cliente utiliza las vías de acceso siguientes:
   - Sistemas UNIX and Linux : `/var/mqm/AMQCLCHL.TAB`
   - Windows: `C:\Program Files\IBM\Websphere MQ\amqclchl.tab`

La primera de las opciones descritas en la lista anterior (utilizando los campos *`ClientConnOffset`* o *`ClientConnPtr`* de MQCNO) sólo 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.

Si recibe un código de retorno MQRC_Q_MGR_NOT_AVAILABLE de la aplicación con un mensaje de error en el archivo de registro de errores de `AMQ9517 -Archivo dañado`, consulte Migración y tablas de definición de canal de cliente (CCDT).

Conceptos relacionados

Tabla de definiciones de canal de cliente

Tareas relacionadas

Configurar conexiones entre el servidor y el cliente

Referencia relacionada

MQSERVER

MQCHLLIB

MQCHLTAB

Conexión de aplicaciones cliente con gestores de colas utilizando variables de entorno

La información de canal de cliente se puede proporcionar a una aplicación que se ejecuta en un entorno de cliente mediante las variables de entorno MQSERVER, MQCHLLIB y MQCHLTAB.

Consulte MQSERVER, MQCHLLIB y MQCHLTAB para obtener detalles de estas variables.

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 Utilización de la estructura MQCNO en una llamada MQCONNX .

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 a qué gestor de colas 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).

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.

Nota: El mismo archivo puede ser utilizado por más de un cliente de IBM WebSphere MQ. Puede acceder a diferentes versiones de este archivo utilizando las variables de entorno MQCHLLIB y MQCHLTAB de IBM WebSphere MQ . Consulte Utilización de variables de entorno de WebSphere MQ para obtener información sobre las variables de entorno.

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 extendido o un cliente Java en WebSphere Application Server, por ejemplo, podría necesitar conectarse a una instancia de gestor de colas predecible. La reconexión automática del cliente no está soportada en WebSphere 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 70 en la página 371 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 con nombre escritos en la CCDT como **QMNAME**(`QM1`) y **QMNAME**(`QMGrp1`), y un grupo en blanco o predeterminado escrito 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.

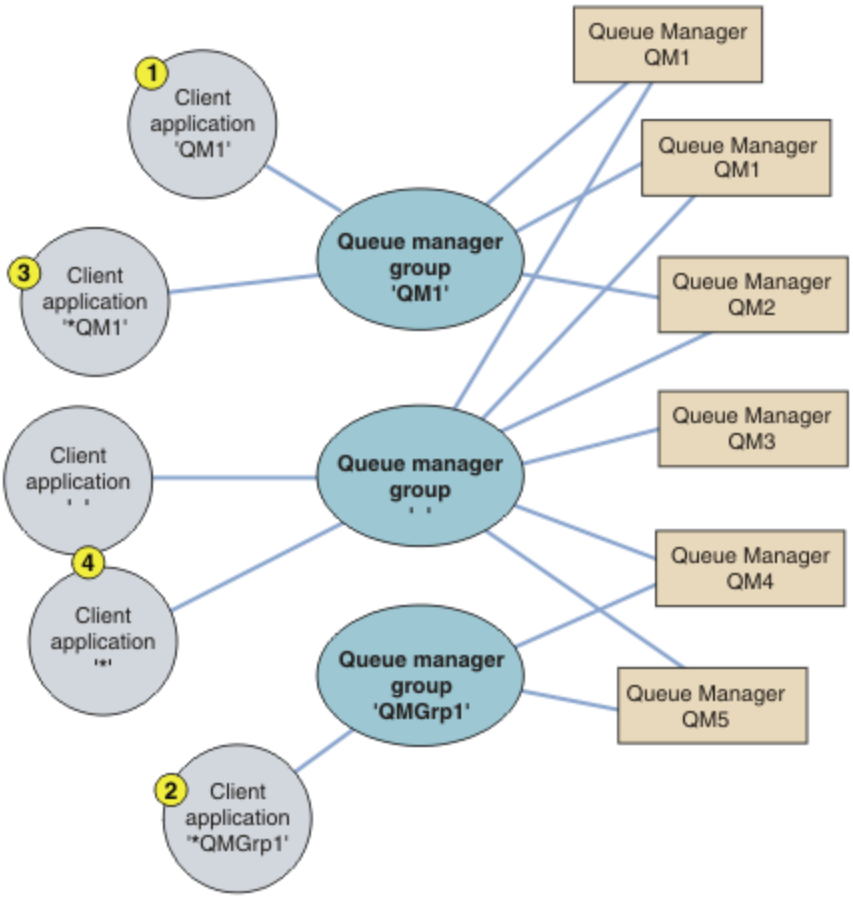


Figura 70. Grupos de gestores de colas

Se describen cuatro ejemplos distintos de uso de esta tabla de conexiones de cliente con la ayuda de las aplicaciones cliente numeradas en Figura 70 en la página 371.

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 de WebSphere MQ selecciona el grupo de gestores de colas coincidente, QM1. El grupo contiene tres canales de conexión y el cliente MQI de WebSphere MQ intenta conectarse a QM1 utilizando cada uno de estos canales a su vez hasta que encuentra un escucha de WebSphere 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 WebSphere MQ selecciona el grupo de gestores de colas coincidente, QMGrp1. Este grupo contiene dos canales de

conexión de cliente y el cliente MQI de WebSphere MQ intenta conectarse a *cualquier* gestor de colas utilizando cada canal a su vez. En este ejemplo, el cliente MQI de WebSphere MQ necesita realizar una conexión satisfactoria; el nombre del gestor de colas al 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, a continuación, intente fusionar las dos tablas de definición de canal de 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 establezca 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.

Equilibrar conexiones de cliente entre gestores de colas, con más clientes conectados a algunos gestores de colas que a 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 cliente MQI de WebSphere MQ aleatoriza las reconexiones entre gestores de colas y las dispersa 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.

Tiene la opción de distribuir la tabla de conexiones de cliente a cada cliente o de colocar la CCDT en un sistema de archivos compartido al que cada cliente pueda hacer referencia. De forma alternativa, utilice la versión programática de la CCDT soportada en la llamada MQI de MQCONNX y llame a un servicio para pasar la CCDT a la aplicación 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.

Conexión con grupos de compartición de colas
Puede conectar la aplicación a un gestor de colas que forme parte de un grupo de compartición de colas. Para ello, utilice 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. Para obtener más información, consulte la sección Conectar un cliente con 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.

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 un cliente MQI de WebSphere MQ selecciona un canal de una CCDT, donde ConnectionAffinity se establece 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 cliente MQI de WebSphere MQ selecciona un canal de una CCDT, donde ConnectionAffinity se establece 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 cliente MQI de WebSphere MQ . (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.

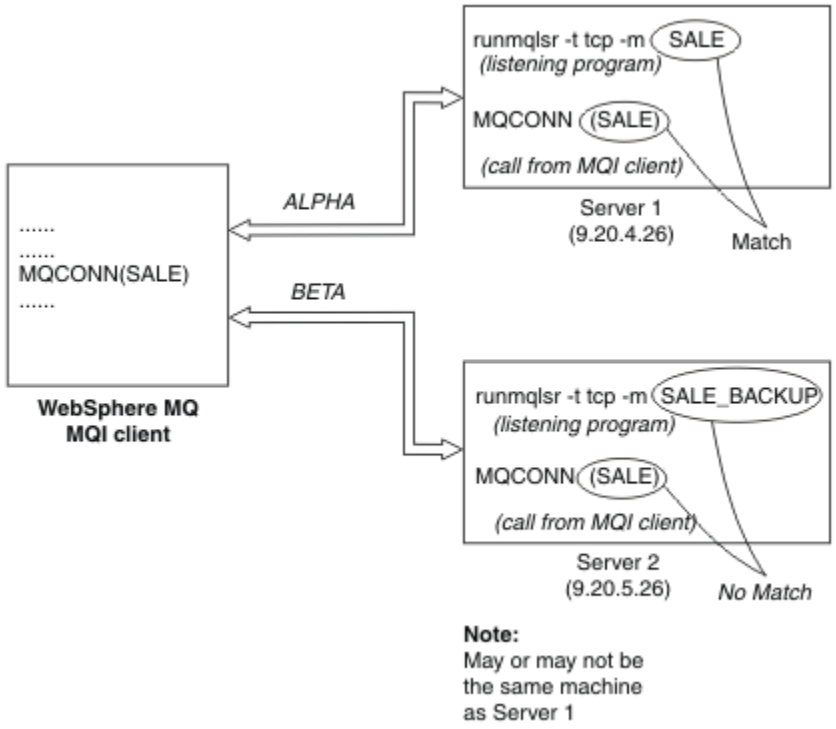


Figura 71. Ejemplo de MQCONN

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 WebSphere MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('WebSphere MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('WebSphere MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Definición de SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')
```

Las definiciones de canal de cliente pueden resumirse como sigue:

Nombre	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALFA	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 realizado un intento de iniciar el canal; *no* se ha realizado correctamente 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 376.

Nota: Si esta aplicación se ejecutara en un entorno que no fuera un cliente MQI de WebSphere MQ y el nombre estuviera 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 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. Esto *no* coincide con el parámetro de llamada MQCONN, que requiere que el nombre del 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. Una vez más, *no* coincide con el parámetro de 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 aplicaciones de WebSphere MQ que se ejecutan en clientes MQI de WebSphere MQ contribuyen a desencadenar 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 transfiere mensajes a una cola desencadenada que es local en 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 cliente Windows y desea enviar una solicitud a un servidor en otro sistema operativo, se debe definir MQAT_WINDOWS_NT; de lo contrario, el otro sistema operativo utilizará sus definiciones predeterminadas y el proceso fallará.

Supervisor desencadenante

El supervisor desencadenante proporcionado por productos que no sonz/OS WebSphere MQ se ejecuta en los entornos de cliente para sistemas UNIX, Linux y Windows .

Para ejecutar el supervisor desencadenante, emita uno de estos comandos:

- Windows Linux UNIX En plataformas Windows, UNIX y Linux :

```
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 elige ejecutar aplicaciones en segundo plano, WebSphere MQ no impone ninguna restricción sobre el número de aplicaciones que se pueden ejecutar.

Para ejecutar la aplicación iniciada en segundo plano en sistemas UNIX and Linux , debe colocar un & (ampersand) al final del *EnvrData* de la definición de proceso.

Aplicaciones CICS (noz/OS)

Un programa de aplicación que no seaz/OS CICS que emita una llamada MQCONN o MQCONNX debe estar definido en CEDA como RESIDENT. Si vuelve a enlazar una aplicación de servidor CICS como cliente, se arriesga a perder el soporte de punto de sincronización.

Un programa de aplicación que no seaz/OS CICS que emita una llamada MQCONN o MQCONNX debe estar definido en CEDA como RESIDENT. Para que el código residente sea lo más pequeño posible, puede enlazar con un programa independiente para emitir la llamada MQCONN o MQCONNX .

Si se utiliza la variable de entorno MQSERVER para definir la conexión de cliente, se debe especificar en CICSENV.CMD .

Las aplicaciones WebSphere MQ se pueden ejecutar en un entorno de servidor WebSphere MQ o en un cliente WebSphere MQ sin cambiar el código. Sin embargo, en un entorno de servidor WebSphere MQ , CICS puede actuar como 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 pierde el soporte de punto de sincronización. MQCMIT y MQBACK deben utilizarse para la aplicación que se ejecuta en un cliente MQI de WebSphere MQ .

Preparación y ejecución de aplicaciones CICS y Tuxedo

Para ejecutar aplicaciones CICS y Tuxedo como aplicaciones cliente, utilice bibliotecas diferentes de 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 CICS y Tuxedo para que se ejecuten como aplicaciones cliente MQI de WebSphere MQ , siga las instrucciones de Configuración de un cliente transaccional extendido.

Sin embargo, tenga en cuenta que la información que se ocupa específicamente de la preparación de aplicaciones CICS y Tuxedo, incluidos los programas de ejemplo proporcionados con WebSphere MQ, presupone que está preparando aplicaciones para que se ejecuten en un sistema servidor WebSphere MQ . Como resultado, la información solo hace referencia a las bibliotecas de WebSphere MQ que están pensadas para su uso en un sistema 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, para aplicaciones escritas en C en AIX, HP-UXo Solaris, utilice la biblioteca libmqic en lugar de libmqm. En sistemas Windows , utilice la biblioteca mqic.lib en lugar de mqm.lib.
- En lugar de las bibliotecas del sistema del servidor que se muestran en Tabla 48 en la página 380, para AIX, HP-UXy Solaris, y Tabla 49 en la página 380, para sistemas Windows , utilice las bibliotecas del sistema cliente equivalentes. Si en estas tablas no se lista una biblioteca del sistema servidor, utilice la misma biblioteca en un sistema cliente.

Tabla 48. Bibliotecas del sistema cliente en AIX, HP-UXy Solaris

Biblioteca para un sistema servidor WebSphere MQ	**Biblioteca equivalente a utilizar en un sistema cliente WebSphere MQ**
libmqmxa	libmqcxa

Tabla 49. Bibliotecas del sistema cliente en sistemas Windows

Biblioteca para un sistema servidor WebSphere MQ	**Biblioteca equivalente a utilizar en un sistema cliente WebSphere 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 WebSphere MQ bajo CICS, normalmente cambia del usuario CICS al ID de usuario de la transacción. Sin embargo, cuando ejecuta una aplicación cliente MQI de WebSphere MQ bajo CICS, conserva la autorización privilegiada de CICS.

Programas de ejemplo CICS y Tuxedo

Programas de ejemplo de CICS y Tuxedo para su uso en sistemas AIX, HP-UX, Solaris y Windows.

La Tabla 50 en la página 380 lista los programas de ejemplo CICS y Tuxedo que se proporcionan para su uso en sistemas cliente AIX, HP-UXy Solaris. La Tabla 51 en la página 381 lista la 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 “El ejemplo de transacción CICS” en la página 121 y “ejemplos de TUXEDO” en la página 160.

Tabla 50. Programas de ejemplo para sistemas cliente AIX, HP-UXy Solaris

Descripción	**Fuente**	**Módulo ejecutable**
Programa CICS	amqscic0.ccs	amqscicc
Archivo de cabecera para el programa CICS	amqscih0.h	-
Programa cliente Tuxedo para transferir mensajes	amqstxpx.c	-

Tabla 50. Programas de ejemplo para sistemas cliente AIX, HP-UXy Solaris (continuación)		
Descripción	**Fuente**	**Módulo ejecutable**
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 51. Programas de ejemplo para sistemas cliente Windows		
Descripción	**Fuente**	**Módulo ejecutable**
Transacción CICS	amqscic0.ccs	amqscicc
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	-

Mensaje de error AMQ5203, tal como se ha modificado para aplicaciones CICS y Tuxedo

Al ejecutar aplicaciones CICS o Tuxedo que utilizan un cliente transaccional extendido, 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 WebSphere MQ se documentan en 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.

Preparación y ejecución de aplicaciones de Microsoft Transaction Server

Para preparar una aplicación MTS para que se ejecute como una aplicación cliente MQI de WebSphere MQ , siga estas instrucciones según convenga para su entorno.

Para obtener información general sobre cómo desarrollar aplicaciones de Microsoft Transaction Server (MTS) que acceden a los recursos de WebSphere MQ , consulte la sección sobre MTS en el Centro de ayuda de WebSphere MQ .

Para preparar una aplicación MTS para que se ejecute como una aplicación cliente MQI de WebSphere MQ , 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 469, pero enlace el componente con la biblioteca mqicxa.lib en vez de mqic.lib.
- Si el componente utiliza las clases C++ de WebSphere MQ , siga las instrucciones de “Creación de programas C++ en Windows” en la página 665 pero enlace el componente con 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 de Visual Basic en Windows” en la página 473, pero cuando defina el proyecto Visual Basic, escriba `MqType=3` en el campo **Argumentos de compilación condicional**,
- Si el componente utiliza WebSphere MQ Automation Classes for ActiveX (MQAX), defina una variable de entorno, GMQ_MQ_LIB, con el valor `mqic32xa.dll` .

  Puede definir la variable de entorno dentro de la aplicación o definirla de forma que su ámbito sea todo el sistema. Sin embargo, la definición a nivel de sistema puede hacer que cualquier aplicación MQAX existente que no defina la variable de entorno desde dentro, se comporte de forma incorrecta.

Preparación y ejecución de aplicaciones JMS de WebSphere MQ

Puede ejecutar aplicaciones JMS de WebSphere MQ en modalidad de cliente, con WebSphere Application Server como gestor de transacciones. Es posible que vea determinados mensajes de aviso.

Para preparar y ejecutar aplicaciones JMS de WebSphere MQ en modalidad de cliente, con WebSphere Application Server como gestor de transacciones, siga las instrucciones de “Utilización de clases de WebSphere MQ para JMS” en la página 729.

Al ejecutar una aplicación cliente JMS de WebSphere MQ , es posible que vea los siguientes mensajes de aviso:

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 WebSphere MQ

Puede ampliar los recursos del gestor de colas utilizando las salidas de usuario, las salidas de API, o los servicios instalables. 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 cómo escribir y compilar salidas y servicios instalables, consulte “Escritura y compilación de salidas y servicios instalables” en la página 383.

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

Escritura y compilación de salidas y servicios instalables

Puede escribir y compilar salidas sin enlazar con ninguna biblioteca de IBM WebSphere MQ en UNIX, Linux y Windows.

Acerca de esta tarea

Windows Linux UNIX Este tema se aplica únicamente a los sistemas Windows, UNIX and Linux . Para obtener información detallada sobre cómo escribir salidas y servicios instalables en otras plataformas, consulte los temas específicos de la correspondiente plataforma.

Si IBM WebSphere MQ se ha instalado en una ubicación no predeterminada, hay que escribir y compilar las salidas sin enlazarlas con ninguna biblioteca de IBM WebSphere MQ.

Puede escribir y compilar salidas en sistemas Windows, UNIX and Linux sin enlazar ninguna de estas bibliotecas IBM WebSphere MQ :

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Las salidas existentes que están enlazadas a estas bibliotecas siguen funcionando, siempre y cuando en los sistemas UNIX and Linux esté instalado IBM WebSphere 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 398.
- 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 408.
- 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 "Escritura de una salida de conversión de datos para WebSphere MQ para Windows" en la página 430 y "Escritura de una salida de conversión de datos para WebSphere MQ en sistemas UNIX and Linux" en la página 426.
  - 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 424.
- 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 432.
- 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 434.
- 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 436.

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 “Programas WebSphere MQ de ejemplo” en la página 98.

3. Compile la salida:
   - No enlace a las bibliotecas de IBM WebSphere MQ .
   - No incluya una RPath incorporada en ninguna biblioteca de IBM WebSphere 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 399.
     - Salidas de canal, salidas de publicación, salidas de carga de trabajo de clúster: “Compilación de programas de salida de canal en sistemas Windows, UNIX and Linux” en la página 423.
     - Salidas de conversión de datos: “Escribir salidas de conversión de datos” en la página 424.
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 WebSphere 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 se coloca una salida en un directorio de instalación específico y en el directorio de vía de acceso predeterminado, la salida del directorio de instalación específico la utiliza la instalación de WebSphere MQ especificada 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 WebSphere MQ installation2 utiliza la salida de /exits/installation2. La instalación de WebSphere 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 393.
   - Salidas de API: “Configurar salidas de API” en la página 403.
   - Salidas de canal: “Configuración de una salida de canal” en la página 424.
   - Salidas de publicación: “Configuración de una salida de publicación” en la página 436.
   - Salidas previas a la conexión: “Stanza PreConnect del archivo de configuración del cliente” en la página 434.

Servicios y componentes instalables para UNIX, Linux y 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.

Las razones principales para proporcionar servicios instalables de WebSphere MQ son:

- Para proporcionarle la flexibilidad de elegir si desea utilizar los componentes proporcionados por los productos WebSphere MQ , o sustituirlos o aumentarlos con otros.
- Para permitir que los proveedores participen, proporcionando componentes que puedan utilizar nuevas tecnologías, sin realizar cambios internos en los productos WebSphere MQ .
- Para permitir que WebSphere MQ aproveche las nuevas tecnologías de forma más rápida y barata y, por lo tanto, proporcione productos antes y a precios más bajos.

Los *servicios instalables* y los *componentes de servicio* forman parte de la estructura del producto WebSphere 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 terceros proporcionar componentes instalables para aumentar o sustituir los proporcionados por los productos WebSphere MQ . En la Tabla 52 en la página 386, se resumen los servicios y componentes que pueden utilizarse.

Tabla 52. 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 **Nota:** Las colas compartidas deben tener su atributo *Scope* establecido en CELL.	• 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.

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 WebSphere MQ para AIX, el componente de servicio de autorización proporcionado, el OAM, realiza todas las funciones disponibles. Para obtener más información, consulte "Interfaz del servicio de autorización" en la página 390. 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 72 en la página 387 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.

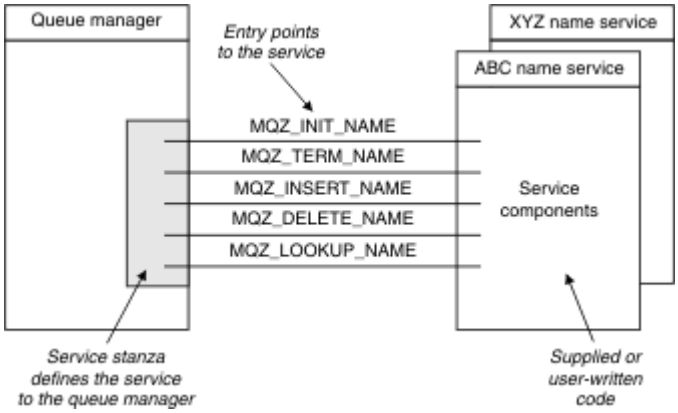


Figura 72. Servicios, componentes y puntos de entrada

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 UNIX 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 ServiceComponentdebe coincidir con el valor de clave de registro de nombre y se define como: HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\Configuration\QueueManager*qmname*\ donde *qmname* es el nombre del gestor de colas.

En los sistemas UNIX 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 de Windows, WebSphere MQ emite una llamada **RegEnumKey** que devuelve los valores en orden alfabético. Por lo tanto, en Windows se llama a los servicios 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 denominado 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 debe establecer el parámetro *Version* en el mismo valor para la inicialización 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 proporcionado con los productos WebSphere MQ se denomina Gestor de autorizaciones sobre objetos (OAM).

De forma predeterminada, el OAM está activo y funciona con los mandatos de control **dspmqaut** (autorización de visualización),**dmpmqaut** (autorización de vuelco) y **setmqaut** (autorización de establecimiento o restablecimiento).

La sintaxis de estos mandatos y cómo utilizarlos se describen en Los mandatos de control.

El OAM funciona con la *entidad* de un principal o grupo.

- En sistemas UNIX 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.
  - el grupo es una colección de principales definida por el sistema UNIX o Linux .
  - 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.
- En los sistemas Windows:
  - el principal es un ID de usuario de Windows o un ID asociado con un programa de aplicación que se ejecuta en nombre de un usuario.
  - el grupo es un grupo de Windows.
  - Las autorizaciones se pueden otorgar o revocar a nivel de principal o de 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 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.

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 ***deben*** tener su atributo *Scope* establecido en 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.

La interfaz de servicio de nombres (NSI) forma parte de la infraestructura de WebSphere 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) sólo se almacena 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 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 WebSphere MQ para Windows, el servicio de autorización proporciona los siguientes puntos de entrada 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 dan soporte al 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 del 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
Busque el nombre del gestor de colas para la cola especificada.

MQZ_INSERT_NAME
Inserte una entrada que contenga el nombre del gestor de colas propietario de 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 53 en la página 393 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 53 en la página 393). 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 53. 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()	NULL
MQZID_DELETE_NAME (Suprimir)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (Buscar)	NULL	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

Configure 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.

1. Añada stanzas al archivo de configuración del gestor de colas para definir el servicio en el gestor de colas y especificar la ubicación del módulo.

   Cada servicio utilizado debe tener una stanza *Service* , que define el servicio para el gestor de colas.

   Para cada componente de un servicio, debe haber una stanza *ServiceComponent* . Dicha stanza identifica el nombre y la ruta del módulo que contiene el código de dicho componente.

   Para obtener más información, consulte “Formato de la stanza Service” en la página 394 y “Formato de la stanza Service Component” en la página 394

   El componente de servicio de autorización, conocido como gestor de autorizaciones sobre objetos (OAM), se proporciona con el producto. Al crear 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 adecuadas 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.

Formato de la stanza Service
La stanza Service contiene el nombre del servicio y el número de puntos de entrada definidos para el servicio.

El formato de la stanza es el siguiente:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

donde:

<service_name>
Nombre del servicio. Este nombre lo define el servicio.

<entries>
Número de puntos de entrada definidos para el servicio. Esto incluye los puntos de entrada de inicialización y terminación.

Formato de stanza de servicio para sistemas Windows
En sistemas Windows , la stanza *Service* incluye un atributo SecurityPolicy .

El formato de la stanza es:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

donde:

<service_name>
Nombre del servicio. Este nombre lo define el servicio.

<entries>
Número de puntos de entrada definidos para el servicio. Esto incluye los puntos de entrada de inicialización y terminación.

<policy>
NTSIDsRequired (el identificador de seguridad de Windows) o Default. Si no especifica NTSIDsRequired, se utiliza el valor Default . Este atributo solo es válido si el valor de Name es AuthorizationService.

Consulte también el tema “Configuración de stanzas de servicio de autorización: sistemas Windows” en la página 395.

Formato de la stanza Service Component

El formato de la stanza de componente de servicio es:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

donde:

<service_name>
Nombre del servicio. Debe coincidir con el valor Name especificado en una stanza Service.

<component_name>
Un nombre descriptivo del componente de servicio. Debe ser exclusivo y contener sólo los caracteres válidos para los nombres de objetos de WebSphere MQ (por ejemplo, nombres de cola). Este nombre aparece en mensajes de operador generados por el servicio. Se le recomienda que utilice un nombre que empiece por una marca comercial de una empresa o una serie de caracteres distintiva similar.

<module_name>
El nombre del módulo que contendrá el código para este componente.

<size>
El tamaño en bytes del área de datos del componente que se pasa al componente en cada llamada. Especifique cero si no se necesitan datos del componente.

Estas dos stanzas pueden ocurrir en cualquier orden y las claves de stanza que contienen también pueden ocurrir en cualquier orden. Para cualquiera de estas stanzas, deben estar presentes todas las claves de stanza. Si se duplica una clave de stanza, se utilizará la última.

Durante el inicio, el gestor de colas procesa en turnos las entradas del componente de servicio que hay en el archivo de configuración. A continuación, carga el módulo del componente especificado, invocando el punto de entrada del componente (que debe ser el punto de entrada para la inicialización del componente) y le pasa un manejador de configuración.

Configuración de stanzas de servicio de autorización: sistemas UNIX and Linux
En sistemas UNIX and Linux, 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 `/var/mqm/qmgrs/QMNAME/qm.ini`.

La stanza *`Service`* y la stanza *`ServiceComponent`* para el componente de autorización predeterminado se añaden a `qm.ini` automáticamente, pero `mqsnoaut`puede alterarlos temporalmente. Cualquier otra stanza *`ServiceComponent`* se debe añadir manualmente.

Por ejemplo, las stanzas siguientes del archivo de configuración del gestor de colas definen dos componentes de servicio de autorización en WebSphere MQ para AIX. *`MQ_INSTALLATION_PATH`* representa el directorio de alto nivel en el que está instalado WebSphere MQ.

```
Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=MQ_INSTALLATION_PATH/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

Figura 73. Stanzas del servicio de autorización UNIX and Linux en el archivo qm.ini

La stanza del componente de servicio, `MQSeries.UNIX.auth.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.

Configuración de stanzas de servicio de autorización: sistemas Windows
En WebSphere MQ para Windows cada gestor de colas tiene su propia stanza en el registro.

La stanza *`Service`* y la stanza *`ServiceComponent`* para el componente de autorización predeterminado se añaden al registro automáticamente, pero se pueden alterar temporalmente utilizando `mqsnoaut`. Cualquier otra stanza *`ServiceComponent`* se debe añadir manualmente.

También puede añadir el atributo SecurityPolicy utilizando los servicios de WebSphere MQ . El atributo SsecurityPolicy sólo se aplica si el servicio especificado en la stanza *Service* es el servicio de autorización, es decir, el OAM predeterminado. El atributo SecurityPolicy permite especificar la política de seguridad para cada gestor de colas. Los valores posibles son:

Default
Especifique Default si desea que la política de seguridad predeterminada entre en vigor. Si no se pasa un identificador de seguridad de Windows (SID de NT) al OAM para un ID de usuario determinado, se intenta obtener el SID adecuado buscando en las bases de datos de seguridad pertinentes.

NTSIDsRequired
Requiere que se pase SID de NT al OAM al efectuar las comprobaciones de seguridad.

Para obtener información sobre el formato de la stanza Service, consulte "Formato de stanza de servicio para sistemas Windows" en la página 394. Para obtener más información general sobre la seguridad, consulte Configuración de la seguridad en sistemas Windows, UNIX and Linux.

La stanza del componente de servicio MQSeries.WindowsNT.auth.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.

Configuración de stanzas de servicio de nombres: sistemas Unix y Linux
Escriba aquí la descripción breve; se utiliza para el primer párrafo y el extracto.

Los ejemplos siguientes de stanzas de archivo de configuración de UNIX and Linux para el servicio de nombres especifican un componente de servicio de nombres proporcionado por la empresa ABC (ficticia).

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

Figura 74. Stanzas del servicio de nombres en qm.ini (para sistemas UNIX and Linux)

Nota: En sistemas Windows , la información de stanza de servicio de nombres se almacena en el registro.

Renovación del gestor de autorizaciones sobre objetos (OAM) después de cambiar la autorización de un usuario
En WebSphere MQ, puede renovar la información del grupo de autorización del OAM inmediatamente después de cambiar la pertenencia al grupo de autorización de un usuario, reflejando los cambios realizados en el nivel del 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 están gestionados por amqzfuma.exe .

Referencia relacionada
REFRESH SECURITY

Escritura y compilación de salidas de API

Las salidas de API le permiten escribir código que modifica el comportamiento de las llamadas de API de WebSphere MQ, por ejemplo MQPUT y MQGET, y luego insertar dicho código inmediatamente antes o después de estas llamadas.

Nota: No soportado en WebSphere MQ para 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. A un nivel superior, puede que desee aplicar estándares o procesos empresariales a un determinado gestor de colas en **todas** las aplicaciones que utilizan dicho gestor. 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:

- En cuanto a la *seguridad*, se 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.
- Respecto a la *flexibilidad*, se puede responder a los cambios rápidos que se produzcan en un entorno empresarial sin cambiar las aplicaciones que dependen de 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.
- En cuanto a la *supervisión* del uso de una cola o un gestor de colas, se puede rastrear el flujo de aplicaciones y mensajes, anotar los errores en las llamadas de API, establecer colas de trazas de auditoría 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 ha escrito un programa de salida y lo ha identificado en WebSphere MQ, el gestor de colas invoca automáticamente el código de salida en los puntos registrados.

Las rutinas de salida de API que se deben ejecutar se identifican en stanzas en sistemas IBM i, Windows, UNIX and Linux . En este tema se tratan las stanzas de los archivos de configuración mqs.ini y qm.ini.

La definición de las rutinas puede realizarse en tres lugares:

1. ApiExitComún, en el archivo mqs.ini , identifica rutinas, para todo WebSphere 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 397 de esta lista).
2. La plantilla ApiExit, en el archivo mqs.ini , identifica rutinas, para todo WebSphere MQ, copiadas en el conjunto local ApiExit(consulte el elemento “3” en la página 397 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 403 determina el orden en el que se ejecutan las rutinas definidas en las stanzas.

Utilización de salidas de API en varias instalaciones de WebSphere MQ

Asegúrese de que las salidas de API escritas para la versión anterior de WebSphere MQ se utilizan para trabajar con todas las versiones porque los cambios realizados en las salidas de la versión 7.1 podrían no funcionar con una versión anterior. Para obtener más información sobre los cambios realizados en las salidas, consulte “Escritura y compilación de salidas y servicios instalables” en la página 383.

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 de WebSphere

MQ correctas que corresponden a la instalación del gestor de colas con el que está asociada la aplicación están enlazadas antes del inicio de la aplicación.

Desarrollo de una salida de API

Se pueden escribir salidas en C para cada llamada de API.

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 WebSphere MQ para un acceso posterior
- MQCLOSE, para cerrar un recurso WebSphere MQ que se había abierto anteriormente 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 los atributos de un recurso de WebSphere MQ que se ha abierto anteriormente para su 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.

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 116.

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 y compilación de salidas y servicios instalables" en la página 383.

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 WebSphere MQ y es específico de 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 una MQ_GET_EXIT, ExitMsgHandle se borra de las propiedades y se rellena con las propiedades especificadas en el campo ExitProperties cuando al registrarse MQ_INIT_EXIT, aparte de 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.

Se proporciona un programa de ejemplo, amqsaem0.c, para ilustrar el uso de los manejadores de mensajes en una salida de API.

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 116. Para plataformas que no sean sistemas Windows , puede encontrar el código de salida de API de ejemplo en *MQ_INSTALLATION_PATH*/samp y la biblioteca compartida compilada y enlazada en *MQ_INSTALLATION_PATH*/samp/bin. Para 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 WebSphere MQ .

Nota para los usuarios:

1. Las directrices sobre la programación de aplicaciones de 64 bits se listan en Estándares de desarrollo en plataformas de 64 bits

Con la introducción de clientes de multidifusión, las salidas de la API y las salidas de conversión de datos se deben poder ejecutar en el extremo del cliente debido a es posible que algunos mensajes no pasen por el gestor de colas. Ahora las siguientes bibliotecas forman parte de los paquetes de cliente así como de los paquetes de servidor:

Tabla 54. Bibliotecas incluidas en los paquetes de cliente y servidor

Sistema operativo	**Bibliotecas**
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb
Linux & HP-UX	32 bits & 64 bits: libmqm.so & libmqm_r.so

Tabla 54. Bibliotecas incluidas en los paquetes de cliente y servidor (continuación)	
Sistema operativo	**Bibliotecas**
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
Solaris	32 bits & 64 bits: libmqm.so

Compilación de salidas de API en sistemas Unix y Linux
Ejemplos de cómo compilar salidas de API en sistemas UNIX y 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 WebSphere MQ.

En 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 -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
    amqsaxe0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

En la plataforma HP-UX Itanium

Aplicaciones de 32 bits

Sin hebras

Compile el código fuente de la salida de API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el código fuente de la salida de API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

Con hebras

Compile el código fuente de la salida de API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el código fuente de la salida de API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

Aplicaciones de 64 bits

Sin hebras

Compile el código fuente de la salida de API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el código fuente de la salida de API

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

Con hebras

Compile el código fuente de la salida de API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el código fuente de la salida de API

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

enLinux

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 \
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

Aplicaciones de 32 bits

Sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

En Solaris

Compile el código fuente de la salida de API emitiendo uno de los mandatos siguientes:

Aplicaciones de 32 bits

Plataforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplicaciones de 64 bits

Plataforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

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 de los ejemplos de Figura 75 en la página 402 o Figura 76 en la página 403 para compilar y enlazar `amqsaxe0.c` en Windows . Los mandatos funcionan con Microsoft Visual Studio 2005, 2008 o 2010. Los ejemplos presuponen que el directorio WebSphere MQ `C:\Program Files\IBM\WebSphere 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 75. Compilar y enlazar `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 76. Compilar y enlazar amqsaxe0.c en Windows de 64 bits

Conceptos relacionados

"El programa de ejemplo de salida de API" en la página 116
La salida de API de ejemplo genera un rastreo MQI para un archivo especificado por el usuario con un prefijo definido en la variable de entorno MQAPI_TRACE_LOGFILE.

Configurar salidas de API

Debe configurar IBM WebSphere 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:

- Utilización de IBM WebSphere MQ Explorer (En Windows y Linux (plataformasx86 y x86-64))
- Utilizando el mandato **amqmdain**, en Windows
- Utilización de los archivos mqs.ini y qm.ini directamente (en sistemas Windows, UNIX and Linux).

  El archivo mqs.ini contiene información relacionada con todos los gestores de colas de un nodo determinado. Puede encontrarlo en el directorio `/var/mqm` en UNIX and Linux y en la WorkPath especificada en la clave `HKLM\SOFTWARE\IBM\WebSphere MQ` en sistemas Windows.

  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:

  En sistemas UNIX and Linux:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  En los sistemas Windows:

  ```
  C:\Program Files\IBM\WebSphere 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:

  – 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

Se define en mqs.ini y en IBM WebSphere MQ Explorer en la página de propiedades IBM WebSphere MQ , bajo Salidas.

Cuando se inicia un gestor de colas, se leen los atributos de esta stanza y luego las salidas de API que están definidas en qm.ini los alteran temporalmente.

ApiExitTemplate

Se define en mqs.ini y en IBM WebSphere MQ Explorer en la página de propiedades IBM WebSphere MQ , bajo 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 WebSphere 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 sustituyen 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 WebSphere 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 WebSphere 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 `auditor`y 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 `tmqp`y 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 WebSphere MQ para 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 424.

Parte de esta información también se aplica a salidas en canales MQI, que conectan clientes MQI de WebSphere MQ a 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 55 en la página 406 muestra los tipos de salida de canal que están disponibles para cada tipo de canal.

Tabla 55. Salidas de canal disponibles para cada tipo de canal

ChannelType	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

Si piensa ejecutar salidas de canal en un cliente, no puede utilizar la variable de entorno MQSERVER. En su lugar, cree y haga referencia a una tabla de definiciones de canal de cliente (CCDT) tal como se describe en Tabla de definiciones de canal 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 77 en la página 407.

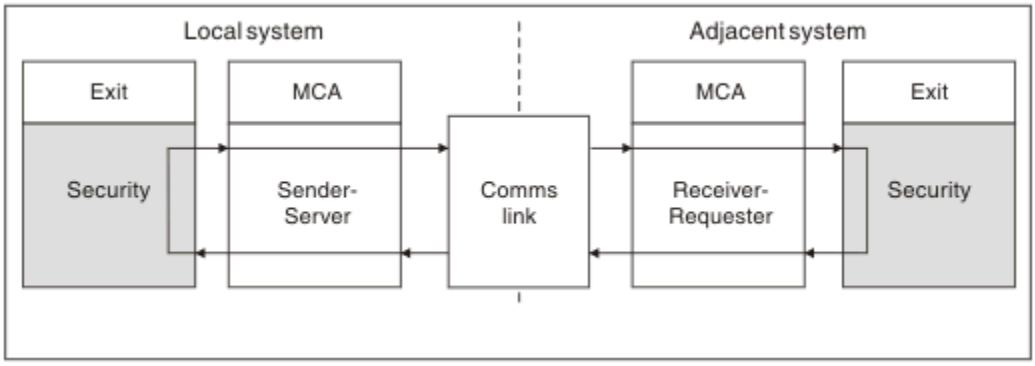


Figura 77. Bucle de salida de seguridad

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 78 en la página 407.

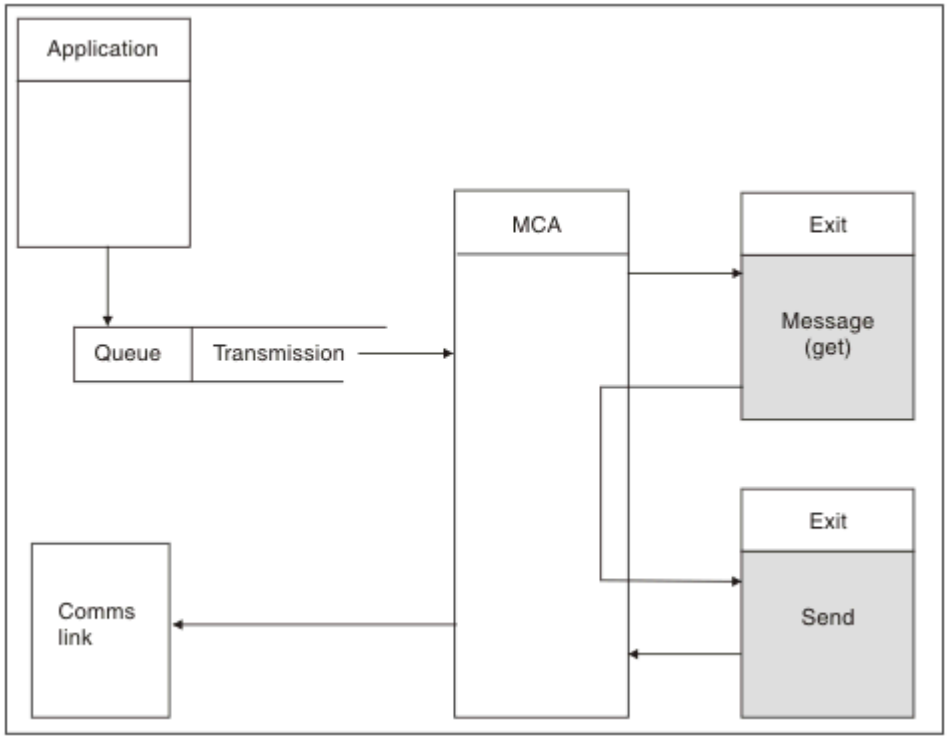


Figura 78. Ejemplo de una salida de envío en el extremo emisor del canal de mensajes

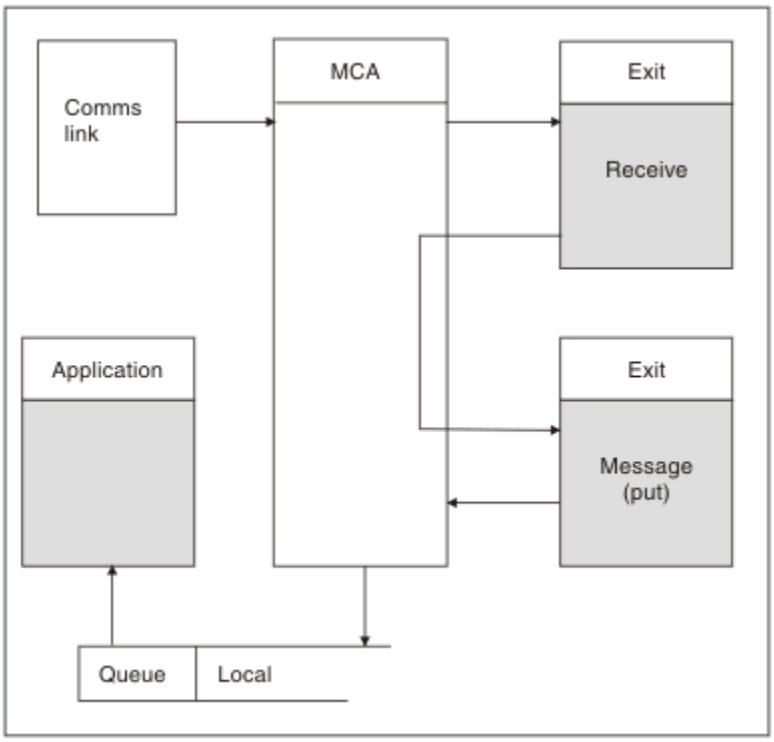


Figura 79. Ejemplo de una salida de recepción en el extremo receptor del canal de mensajes

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 79 en la página 408. 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 ha enlazado con MQM.LIB y no especifica un nombre de gestor de colas en la llamada MQCONN, la salida intenta conectarse al gestor de colas predeterminado del sistema. Si la salida se ha enlazado con MQM.LIB y especifica el nombre del gestor de colas que se ha pasado a la salida a través del campo QMgrName de MQCD, la salida intenta conectarse a ese 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 WebSphere 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 en la estructura de datos MQCD, el nuevo valor es ignorado por el proceso de canal IBM WebSphere MQ . 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.

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:
  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 405 para obtener detalles de la llamada.

Escritura de programas de salida de canal en sistemas Windows, UNIX and Linux
Puede utilizar la siguiente información como ayuda para escribir programas de salida de canal para sistemas Windows, UNIX and Linux .

Siga las instrucciones descritas en “Escritura y compilación de salidas y servicios instalables” en la página 383. Utilice la siguiente información específica de salida de canal, si procede:

La salida debe estar escrita 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 80 en la página 410 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 80. Código fuente de ejemplo para una salida de canal

Al escribir salidas de canal para Windows utilizando Visual C++, debe escribir su propio archivo DEF . En Figura 81 en la página 410 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 408.

```
EXPORTS
ChannelExit
```

Figura 81. Archivo DEF de ejemplo para Windows

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 WebSphere MQ. El soporte SSL y TLS también lo proporciona IBM WebSphere MQ para autenticar a los usuarios y proporcionar comprobaciones de cifrado e integridad de datos para los datos. Para obtener más información sobre SSL y TLS, consulte WebSphere MQ support for SSL and TLS. 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.

Para salidas de seguridad escritas antes de IBM WebSphere MQ Version 7.1 cabe señalar que las versiones anteriores de IBM WebSphere MQ consultaban el proveedor de sockets seguros subyacente (por ejemplo, GSKit) para determinar el Nombre distinguido del sujeto (SSLPEER) y el Nombre distinguido del emisor (SSLCERTI) del certificado del asociado remoto. En IBM WebSphere MQ Version 7.1 se ha añadido soporte para una serie de nuevos atributos de seguridad. Para acceder a estos atributos IBM WebSphere MQ Version 7.1 obtiene la codificación DER del certificado y lo utiliza para determinar el DN de sujeto y de emisor. 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

En IBM WebSphere MQ Version 7.1, también se incluye un atributo SERIALNUMBER en el nombre distinguido (DN) del sujeto y contiene el número de serie del certificado del asociado remoto. Además, algunos atributos DN se devuelven en una secuencia diferente que en releases anteriores. Por consiguiente, la composición de los campos SSLPEER y SSLCERTI se modifican en Version 7.1 con respecto a releases anteriores y, por lo tanto, se recomienda que se examinen y actualicen todas las salidas de seguridad o las aplicaciones dependientes de estos campos.

Los filtros de nombre de igual existentes de WebSphere MQ especificados a través del campo SSLPEER de una definición de canal no se ven afectados y seguirán funcionando de la misma forma que en releases anteriores. Esto se debe a que el algoritmo de coincidencia de nombres de igual de WebSphere MQ se ha actualizado para procesar filtros SSLPEER existentes sin necesidad de modificar las definiciones de canal. Este cambio probablemente afectará a las salidas de seguridad y aplicaciones que dependen de los valores de DN de sujeto y DN de emisor devueltos por la interfaz de programación PCF.

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 82 en la página 412 a Figura 85 en la página 414 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 82 en la página 412:

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.

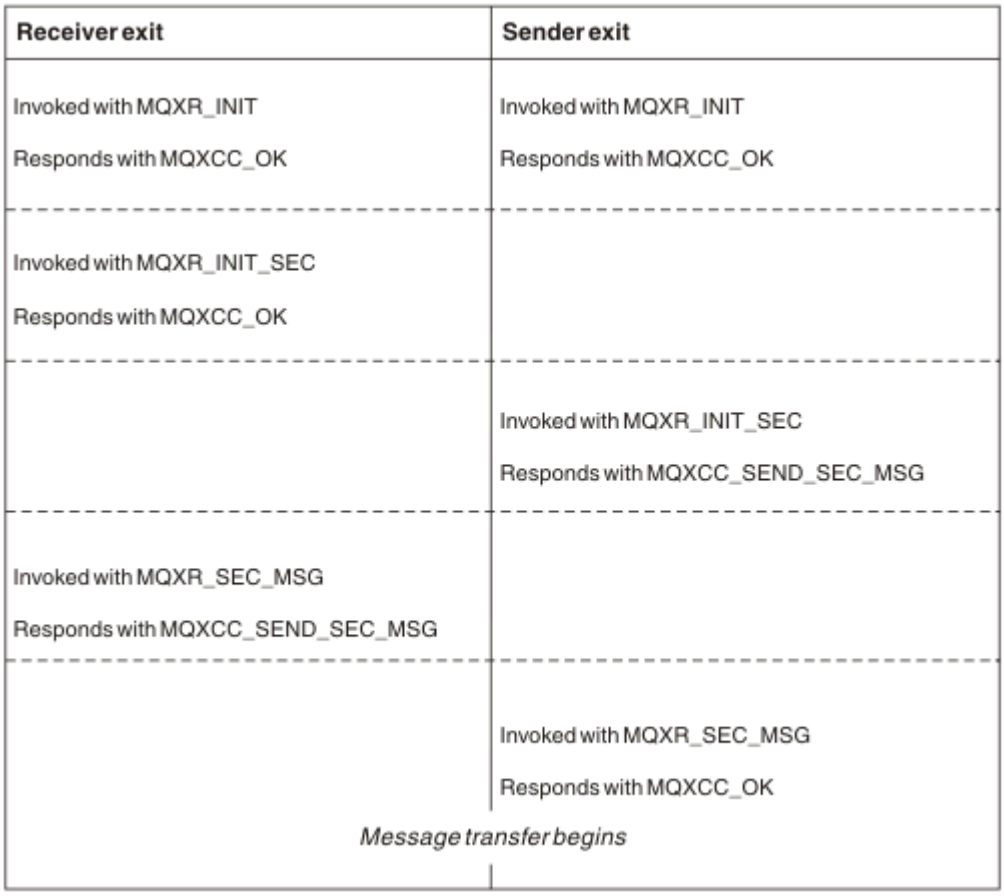


Figura 82. Intercambio con acuerdo iniciado por el emisor

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 83. 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 84. 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 85. 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. En IBM i, los programas de salida de seguridad que se han compilado con "Utilizar autorización adoptada" (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 SSL en el que el otro extremo del canal proporciona un certificado, la salida de seguridad recibe el nombre distinguido del asunto de este certificado en el campo MQCD al que accede SSLPeerNamePtr y el nombre distinguido del emisor en el campo MQCXP al que accede SSLRemCertIssNamePtr. Este nombre puede servir:
   - Para restringir el acceso a través del canal SSL.
   - para cambiar MQCD.MCAUserIdentifier en función del nombre.

Conceptos relacionados

Registros de autenticación de canal

Conceptos SSL (Secure Sockets Layer) y TLS (Transport Layer Security)

Desarrollo de una salida de seguridad

Se puede escribir una salida de seguridad utilizando el código esqueleto de salida de seguridad.

Figura 86 en la página 415 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 86. Código de esqueleto de una salida de seguridad

El punto de entrada MQStart estándar de WebSphere MQ debe existir, pero no es necesario para realizar 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
WebSphere MQ para Windows proporciona una salida de seguridad que proporciona autenticación para canales WebSphere MQ utilizando la interfaz de programación de servicios de seguridad (SSPI). El SSPI proporciona los recursos de seguridad integrados de Windows.

Esta salida de seguridad es para el cliente WebSphere MQ y el servidor WebSphere MQ .

Los paquetes de seguridad se cargan desde security.dll o secur32.dll. Estas DLL se suministran con el sistema operativo.

La autenticación unidireccional se proporciona en Windows, utilizando los servicios de autenticación 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 sistemas Windows" en la página 173

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 emisión se invoca inmediatamente antes de la transmisión, con *ExitReason* establecido en MQXR_XMIT, se reservan *ExitSpace* bytes en el almacenamiento intermedio 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.

¿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. WebSphere 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.

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

A la salida de mensajes se le pasa un almacenamiento intermedio de agente que contiene la cabecera de cola de transmisión, MQXQH, y el texto del mensaje de aplicación tal como se recupera de la cola. (El formato de MQXQH se proporciona en MQXQH.) Si utiliza mensajes de referencia; es decir, mensajes que contienen sólo 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 272 y “Ejecución de los ejemplos de mensajes de referencia” en la página 145 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. En sistemas UNIX , si proporciona una salida de mensajes por cualquier motivo, la conversión automática de los ID de usuario a caracteres en minúsculas no funciona. Consulte Seguridad de objetos en sistemas UNIX and Linux.
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, el canal confirma o restituye las llamadas realizadas bajo punto de sincronización 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, es posible emitir llamadas MQI de punto de sincronización 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 nombres de formato que empiecen por los tres caracteres 'MQH' (por ejemplo, "MQHRF ") que no se mencionan de otro modo

Cómo se procesan las cabeceras

El MCA lee el parámetro de formato de cada cabecera de WebSphere MQ . 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 el formato es el nombre de un tipo de cabecera elegible para la conversión de datos de WebSphere 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 WebSphere MQ para 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.

Se puede llamar a la salida de definición automática de canal en todas las plataformas excepto z/OS cuando se recibe una solicitud para iniciar un canal receptor o de conexión con el 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 WebSphere MQ para 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 direccionado por MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr y ReceiveUserDataPtr, en lugar de MsgExit, MsgUserData, SendExit, Los propios campos de datos SendUser, ReceiveExit y ReceiveUser.

Para obtener más información, consulte Definición automática de canales.

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.

Compilación de programas de salida de canal en sistemas Windows, UNIX and Linux

Utilice los ejemplos siguientes como ayuda para compilar programas de salida de canal para sistemas Windows, UNIX and Linux .

Windows

Windows

El compilador y el mandato de enlazador para 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 UNIX y Linux

Linux UNIX

En estos ejemplos, `exit` es el nombre de biblioteca y `ChannelExit` es el nombre de función. En AIX el archivo de exportación se denomina `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.

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
```

Mandatos de compilador y enlazador de ejemplo para salidas de canal en HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

Mandatos de compilador y enlazador de ejemplo para salidas de canal en plataformas Linux donde el gestor de colas es de 32 bits:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Mandatos de compilador y enlazador de ejemplo para salidas de canal en plataformas Linux donde el gestor de colas es de 64 bits:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Mandatos de compilador y enlazador de ejemplo para salidas de canal en Solaris:

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

En el cliente, se puede utilizar una salida de 32 bits o de 64 bits. Esta salida se debe enlazar a mqic_r.

En AIX, todas las funciones a las que llama IBM WebSphere MQ deben exportarse. 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 WebSphere MQ ; consulte MQCD o Mandatos de script (MQSC) para obtener 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. Puesto que WebSphere MQ debe ser capaz de comprender el contenido del MQMD independientemente de la plataforma en la que se crea, el sistema lo convierte automáticamente.

No obstante, los datos de la aplicación no se convierten automáticamente. Si se están intercambiando datos de tipo carácter entre plataformas en las que 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 contiene información sobre el recurso de salida de conversión de datos que WebSphere MQ proporciona cuando los datos de aplicación no están en un formato incorporado.

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 WebSphere 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* en MQMD en el juego de caracteres y la codificación necesarios. Si la aplicación utiliza el mismo juego de caracteres y 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 distintas codificaciones de máquina y CCSID deben ajustarse a la interfaz de conversión de datos (DCI) de WebSphere MQ .

Con la introducción de clientes de multidifusión, las salidas de la API y las salidas de conversión de datos se deben poder ejecutar en el extremo del cliente debido a es posible que algunos mensajes no pasen por el gestor de colas. Ahora las siguientes bibliotecas forman parte de los paquetes de cliente así como de los paquetes de servidor:

Tabla 56. Bibliotecas incluidas en los paquetes de cliente y servidor

Sistema operativo	**Bibliotecas**
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb
Linux & HP-UX	32 bits & 64 bits: libmqm.so & libmqm_r.so
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
Solaris	32 bits & 64 bits: libmqm.so

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 cumple lo siguiente:

- 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 con el mensaje no es MQFMT_NONE.
- El *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 al 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.

Escritura de una salida de conversión de datos para WebSphere MQ en sistemas UNIX and Linux

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos para WebSphere MQ en sistemas UNIX and Linux .

Siga estos pasos:

1. Nombre el formato del mensaje. El nombre debe caber en el campo *Format* del MQMD y estar en mayúsculas, por ejemplo, MYFORMAT. El nombre de *Format* no debe tener espacios en blanco iniciales. Los espacios en blanco finales se ignoran. El nombre del objeto no debe tener más de ocho caracteres no en blanco, porque *Format* sólo 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 426. 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 WebSphere MQ para AIX, también se proporciona un archivo de exportación de esqueleto denominado 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 que está instalado WebSphere 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 426.
   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 427. 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 426 anterior.
8. Compile la salida como una biblioteca compartida utilizando MQStart como punto de entrada. Para hacerlo, consulte "Compilación de una salida de conversión de datos en sistemas UNIX and Linux" en la página 427.
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 WebSphere MQ deben compilarse 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.

Compilación de una salida de conversión de datos en sistemas UNIX and Linux
Ejemplos de cómo compilar una salida de conversión de datos en sistemas UNIX 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 WebSphere MQ.

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 -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
    MYFORMAT.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Plataforma HP-UX Itanium

Compile y enlace el código fuente de la salida ejecutando uno de los siguientes conjuntos de comandos:

Aplicaciones de 32 bits

Sin hebras

Compile el código fuente de la salida:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el objeto de la salida:

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

Con hebras

Compile el código fuente de la salida:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el objeto de la salida:

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
    -lpthread
rm MYFORMAT.o
```

Aplicaciones de 64 bits

Sin hebras

Compile el código fuente de la salida:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el objeto de la salida:

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

Con hebras

Compile el código fuente de la salida:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Enlace el objeto de la salida:

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

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 \
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Aplicaciones de 32 bits

Sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Aplicaciones de 64 bits

Sin hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Con hebras

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Solaris

Compile el código fuente de la salida emitiendo uno de los comandos siguientes:

Aplicaciones de 32 bits

Plataforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplicaciones de 64 bits

Plataforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64  -lsocket -lnsl -ldl
```

Escritura de una salida de conversión de datos para WebSphere MQ para Windows

Información sobre los pasos a tener en cuenta al escribir programas de salida de conversión de datos para WebSphere MQ para Windows.

Siga estos pasos:

1. Nombre el formato del mensaje. El nombre debe caber en el campo *Format* del MQMD. El nombre de *Format* no debe tener espacios en blanco iniciales. Los espacios en blanco finales se ignoran. El nombre del objeto no debe tener más de ocho caracteres no en blanco, porque *Format* sólo 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 WebSphere 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 430.

   amqsvfc0.c se encuentra en *MQ_INSTALLATION_PATH*\Tools\C\Samples , donde *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ . (El directorio de instalación predeterminado es `C:\Program Files\IBM\WebSphere 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 430.
   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 431. 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 430.
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 donde está instalado WebSphere MQ .
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 WebSphere 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\MQSeries\CurrentVersion\Configuration\ClientExitPa
   th\
   ```

   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 empaquetadas, todas las aplicaciones WebSphere MQ deben compilarse 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.

Archivos de carga de salida y conmutación en sistemas operativos Windows

Los procesos del gestor de colas IBM WebSphere MQ for Windows Version 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 WebSphere MQ . El uso de estos atributos garantiza que se pueda encontrar la biblioteca adecuada. Si se utiliza una salida en un clúster de WebSphere MQ , esto también garantiza que se pueda encontrar la biblioteca adecuada en un 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 archivos	**Aplicaciones de 32 bits**	**Aplicaciones de 64 bits**
salida cruzada de API	32 bits	32 bits y 64 bits
Salida de conversión de datos	32 bits	64 bits
Salidas de canal servidor (todos los tipos)	32 bits	32 bits
Salidas de canal cliente (todos los tipos)	32 bits	64 bits
Salida de servicio instalable	32 bits	32 bits
Módulo de rastreo de servicio	32 bits	32 bits y 64 bits
Salida de WLM de clúster	32 bits	32 bits
Salida de direccionamiento Pub/Sub	32 bits	32 bits
Archivos de carga de conmutación de base de datos	32 bits	32 bits y 64 bits
Bibliotecas XA de gestor de transacciones externo	32 bits	64 bits

Referencia a las definiciones de conexión utilizando una salida de preconexión desde un depósito

Los clientes MQI de WebSphere MQ se pueden configurar para buscar un repositorio para obtener definiciones de conexión utilizando una biblioteca de salida previa a la conexión.

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. Puesto que es difícil gestionar y administrar diversas aplicaciones cliente que hacen referencia al archivo CCDT, un enfoque flexible consiste en almacenar las definiciones de cliente en un repositorio global 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 WebSphere 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*/
```

Stanza PreConnect del archivo de configuración del cliente

Utilice la stanza PreConnect para configurar la salida PreConnect en el archivo `mqclient.ini`.

Los siguientes atributos se pueden incluir en la stanza PreConnect:

Data=*< URL>*
URL del depósito en el que se almacenan las definiciones de conexión. Por ejemplo, cuando se utiliza un servidor LDAP:

Datos = ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com

Function=*<myFunc>*
Nombre del punto de entrada funcional a la biblioteca que contiene el código de salida de Preconnect.

La definición de función se adhiere al prototipo de salida de Preconnect MQ_PRECONNECT_EXIT.

La longitud máxima de este campo es MQ_EXIT_NAME_LENGTH.

Module=*< pi_amqlda>*
El nombre del módulo que contiene el código de salida de la API.

Si este campo contiene el nombre de vía de acceso completo del módulo, se utiliza tal cual.

Sequence=*< número_secuencia>*
La secuencia en la que se llama a esta salida en relación con otras salidas. 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 continuos; una secuencia de 1, 2, 3 tiene el mismo resultado que una secuencia de 7, 42, 1096. Este atributo es un valor numérico sin signo.

Se pueden definir varias stanzas PreConnect dentro del archivo `mqclient.ini`. El orden de proceso de cada salida está determinado por el atributo Sequence de la stanza.

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.

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 y compilación de salidas y servicios instalables” en la página 383 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.

WebSphere 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.

Es posible que cada conexión de WebSphere MQ que requiera la salida de publicación no pueda 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 deWebSphere 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 y compilación de salidas y servicios instalables" en la página 383.

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 WebSphere 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 sistemas UNIX y Linux , cree una stanza denominada `PublishSubscribe`. La stanza `PublishSubscribe` tiene los atributos siguientes:

PublishExitPath=[path]|*module_name*
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=*function_name*
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`.

PublishExitData=*string*
Si el gestor de colas invoca una salida de publicación, pasa una estructura `MQPSXP` como entrada. Los datos especificados utilizando 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. Sin embargo, para que pueda proporcionar su propio programa de salida de usuario para adaptar la gestión de carga de trabajo, WebSphere 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 mientras se transfiere el mensaje y se restituye el mensaje, se selecciona un nuevo gestor de colas de destino.

En plataformas que no sean z/OS, el gestor de colas cargará la nueva salida de carga de trabajo de clúster la próxima vez que se inicie 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 que no sonz/OS , si utiliza CLWLMode=FAST, cada proceso de 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

Para plataformas que no sean z/OS, las salidas de carga de trabajo de 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 y compilación de salidas y servicios instalables” en la página 383, y utilice el programa de ejemplo de la sección “Salida de carga de trabajo de clúster de ejemplo” en la página 437 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 408.

Configuración de salidas de carga de trabajo de clúster

Las salidas de carga de trabajo de clúster se denominan en la definición del 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)
```

Salida de carga de trabajo de clúster de ejemplo

WebSphere 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.

En plataformas distintas de z/OS

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 57. Ubicación del programa de salida de carga de trabajo de clúster de ejemplo (no z/OS)	
Plataforma	**Vía de acceso de archivo**
AIX, HP-UX, Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Windows	*MQ_INSTALLATION_PATH*\Tools\c\Samples

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere 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. Proporcione 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:

En sistemas UNIX and Linux:

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

En Windows:

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

Ahora, en lugar de utilizar el algoritmo de gestión de carga de trabajo proporcionado, WebSphere MQ llama a esta salida para direccionar todos los mensajes al gestor de colas elegido.

Creación de una aplicación IBM WebSphere MQ

Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

Creación de la aplicación en AIX

Las publicaciones de AIX describen cómo crear aplicaciones ejecutables a partir de los programas que escribe.

En este tema se describen las tareas adicionales y los cambios en las tareas estándar que debe realizar al crear aplicaciones WebSphere 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 WebSphere MQ para AIX varían con el lenguaje de programación en el que está escrito 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 WebSphere MQ para AIX para el idioma que está utilizando. Familiarícese con el contenido de estos archivos. Consulte la sección “IBM WebSphere MQ archivos de definición de datos” en la página 82 para obtener una descripción completa.

Cuando ejecute aplicaciones de cliente o servidor con hebras, establezca la variable de entorno AIXTHREAD_SCOPE=S.

Preparación de programas C en AIX

Este tema contiene información sobre el enlace de bibliotecas necesarias para preparar programas C en AIX.

Los programas C precompilados se proporcionan en el directorio *MQ_INSTALLATION_PATH*/samp/bin . Utilice el compilador ANSI y ejecute los mandatos siguientes. Para obtener más información acerca de cómo programar aplicaciones de 64 bits, consulte la sección Estándares de codificación en las plataformas de 64 bits.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Para aplicaciones de 32 bits:

```
$ xlc_r -o amqsput_32 amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

donde `amqsput0` es un programa de ejemplo.

Si utiliza el compilador VisualAge C/C++ para programas C++, debe incluir la opción `-q namemangling=v5` para obtener todos los símbolos WebSphere MQ resueltos al enlazar las bibliotecas.

Si desea utilizar los programas en una máquina que solo tiene instalado el cliente MQI de WebSphere MQ para AIX , vuelva a compilar los programas para enlazarlos con la biblioteca de cliente (`-lmqic`) en su lugar.

Enlazar bibliotecas

Necesita las bibliotecas siguientes:

- Enlace los programas con la biblioteca adecuada proporcionada por WebSphere 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 WebSphere MQ de hebra simple a partir de una sola unidad de compilación, ejecute los mandatos siguientes.

  Para aplicaciones de 32 bits:

```
  $ xlc_r -o amqsputc_32_r amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm_r
```

  donde `amqsput0` es un programa de ejemplo.

  Si desea utilizar los programas en una máquina que solo tiene instalado el cliente MQI de WebSphere MQ para AIX , vuelva a compilar los programas para enlazarlos con la biblioteca de cliente (`-lmqic`) en su lugar.

  Nota:

  1. Si está escribiendo un servicio instalable (consulte Administración 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.
  2. Si está generando una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como IBM TXSeries, Encinao BEA Tuxedo, debe enlazar con las bibliotecas

`libmqmxa.a` (o `libmqmxa64.a` si el 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.

3. Es necesario enlazar aplicaciones de confianza a las bibliotecas de WebSphere MQ con hebras. Sin embargo, solo se puede conectar una hebra de una aplicación de confianza en WebSphere MQ en sistemas UNIX and Linux a la vez.
4. Debe enlazar las bibliotecas de WebSphere MQ antes que cualquier otra biblioteca de producto.

Preparación de programas en 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 se instala IBM WebSphere 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 utilizando IBM Conjunto COBOL para AIX

Se proporcionan ejemplos de programas en COBOL en IBM WebSphere 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 \
-I<COBCPY>
```

Aplicación cliente sin hebras de 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-I<COBCPY>
```

Aplicación de servidor con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

Aplicación cliente con hebras de 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

Aplicación de servidor sin hebras de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -I<COBCPY>
```

Aplicación cliente sin hebras de 64 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -I<COBCPY>
```

Aplicación de servidor con hebras de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

Aplicación cliente con hebras de 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

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>
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.

Preparación de programas de aplicación CICS en AIX

Utilice esta información al preparar programas CICS en AIX.

Los módulos de conmutador XA se proporcionan para permitirle enlazar CICS con IBM WebSphere MQ:

Tabla 58. 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 precompilada del IBM WebSphere MQ archivo de carga conmutada *amqzsc* , que se proporciona con el producto.

Enlace siempre las transacciones C con la biblioteca IBM WebSphere MQ de hebras seguras *libmqm_r.a.*, y las transacciones COBOL con la biblioteca COBOL *libmqmcb_r.a.*.

Puede encontrar más información sobre las transacciones CICS de soporte en Administración.

Soporte de TXSeries CICS

IBM WebSphere MQ en AIX da soporte a TXSeries CICS utilizando la interfaz XA. Asegúrese de que las aplicaciones CICS estén enlazadas a la versión con hebras de las bibliotecas de IBM WebSphere 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 WebSphere MQ que se carguen en la misma región CICS en C o COBOL. No puede realizar una combinación de llamadas MQI de C y COBOL en la misma región 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 IBM Conjunto COBOL para AIX

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere 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  \
                   -IMQ_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 COBOL de CICS utilizando Micro Focus COBOL

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.

Para utilizar Micro Focus COBOL, siga estos pasos:

1. Añada el módulo de bibliotecas de tiempo de ejecución de IBM WebSphere MQ COBOL a la biblioteca de tiempo de ejecución utilizando el mandato siguiente:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Nota: Con `cicsmkcobol`, IBM WebSphere 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 WebSphere 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 del método de lenguaje Micro Focus COBOL y permite que la biblioteca COBOL de tiempo de ejecución de CICS invoque IBM WebSphere MQ en sistemas UNIX and Linux.

Nota: Ejecute `cicsmkcobol` solo cuando instale uno de los productos siguientes:

- Versión o release nuevo de Micro Focus COBOL
- Nueva versión o release de CICS para AIX
- Versión o release nuevo de cualquier producto de base de datos soportado (solo para las transacciones COBOL)
- Nueva versión o release de IBM WebSphere 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 se instala IBM WebSphere MQ.

Cree programas C de CICS utilizando los recursos estándar de CICS :

1. Exporte ***una*** de las variable 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 de ejemplo C CICS

AMQSCIC0.CCS proporciona código fuente C de ejemplo para una transacción AIX IBM WebSphere 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.

Creación de la aplicación en HP Integrity NonStop Server

Esta información describe las tareas adicionales y los cambios en las tareas estándar que hay que realizar al crear aplicaciones cliente de IBM WebSphere MQ para HP Integrity NonStop Server para que ejecuten bajo HP Integrity NonStop Server.

Se soportan C, COBOL y pTAL.

Cabeceras y bibliotecas públicas de OSS y Guardian

Se proporcionan lista de cabeceras y bibliotecas públicas de OSS y Guardian. A continuación figuran las cabeceras, el ejecutable público y las bibliotecas de importación públicas de OSS, las cabeceras y los ejecutables públicos y las bibliotecas de importación públicas de Guardian.

"Cabeceras de OSS" en la página 445

"Bibliotecas ejecutables públicas y bibliotecas de importación públicas de OSS" en la página 445

“Cabeceras de Guardian” en la página 446

“Bibliotecas ejecutables públicas y bibliotecas de importación públicas de Guardian” en la página 447

Cabeceras de OSS

Tabla 59. Cabeceras de OSS

Objeto	Ubicación	Descripción
cmqbc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqfc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqec.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqpsc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqxc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqzc.h	<mqinstall>/inc	Cabecera de lenguaje C de IBM WebSphere MQ (OSS)
cmqcobol.cpy	<mqinstall>/inc	Libro de copias en lenguaje COBOL de IBM WebSphere MQ (OSS)
cmqbt.tal	<mqinstall>/inc	Cabecera pTAL de IBM WebSphere MQ (OSS)
cmqcft.tal	<mqinstall>/inc	Cabecera pTAL de IBM WebSphere MQ (OSS)
cmqpst.tal	<mqinstall>/inc	Cabecera pTAL de IBM WebSphere MQ (OSS)
cmqt.tal	<mqinstall>/inc	Cabecera pTAL de IBM WebSphere MQ (OSS)
cmqxt.tal	<mqinstall>/inc	Cabecera pTAL de IBM WebSphere MQ (OSS)

Bibliotecas ejecutables públicas y bibliotecas de importación públicas de OSS

Tabla 60. Bibliotecas ejecutables públicas y bibliotecas de importación públicas de OSS

Objeto	Ubicación	Descripción
libmqic.so	<mqinstall>/bin	Biblioteca ejecutable pública de IBM WebSphere MQ (sin hebras OSS)
libmqic_r.so	<mqinstall>/bin	Biblioteca ejecutable pública de IBM WebSphere MQ (con múltiples hebras OSS)

Tabla 60. Bibliotecas ejecutables públicas y bibliotecas de importación públicas de OSS (continuación)		
Objeto	**Ubicación**	**Descripción**
libmqic.so	<mqinstall>/lib	Biblioteca de importación pública de IBM WebSphere MQ (OSS sin hebras)
libmqic_r.so	<mqinstall>/lib	Biblioteca de importación pública de IBM WebSphere MQ (con múltiples hebras OSS)
mqicb	<mqinstall>/lib	Biblioteca de importación pública de IBM WebSphere MQ para COBOL (OSS)

Cabeceras de Guardian

Tabla 61. Cabeceras de Guardian		
Objeto	**Ubicación**	**Descripción**
cmqbch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqfch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqech	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqpsch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqxch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqzch	<mqinstall>/inc/G	Cabecera de lenguaje C de IBM WebSphere MQ (Guardian)
cmqcobol	<mqinstall>/inc/G	Libro de copias en lenguaje COBOL de IBM WebSphere MQ (Guardian)
cmqbt	<mqinstall>/inc/G	Cabecera pTAL de IBM WebSphere MQ (Guardian)
cmqcft	<mqinstall>/inc/G	Cabecera pTAL de IBM WebSphere MQ (Guardian)
cmqpst	<mqinstall>/inc/G	Cabecera pTAL de IBM WebSphere MQ (Guardian)
cmqt	<mqinstall>/inc/G	Cabecera pTAL de IBM WebSphere MQ (Guardian)
cmqxt	<mqinstall>/inc/G	Cabecera pTAL de IBM WebSphere MQ (Guardian)

Bibliotecas ejecutables públicas y bibliotecas de importación públicas de Guardian

Tabla 62. Bibliotecas ejecutables públicas y bibliotecas de importación públicas de Guardian		
Objeto	**Ubicación**	**Descripción**
mqic	<mqinstall>/bin/G	Biblioteca ejecutable pública de IBM WebSphere MQ (Guardian)
mqicb	<mqinstall>/lib/G	Biblioteca de importación pública de IBM WebSphere MQ para COBOL (Guardian)

Preparación de programas C en HP Integrity NonStop Server

Este tema contiene información que debe tenerse en cuenta cuando se preparan programas C en HP Integrity NonStop Server, junto con ejemplos de los mandatos que se utilizan cuando se crean aplicaciones, cuando se utiliza el compilador C de OSS, y cuando se utiliza el compilador C de Guardian.

Los programas C precompilados se proporcionan en el directorio `MQ_INSTALLATION_PATH/opt/mqm/samp/bin` . Para crear un ejemplo a partir del código fuente, utilice el compilador c89.

Debe enlazar sus programas con la biblioteca adecuada que proporciona IBM WebSphere MQ. En la tabla siguiente, se listan las bibliotecas que debe enlazar cuando se preparan programas C en HP Integrity NonStop Server.

Tabla 63. . Bibliotecas de enlace de HP Integrity NonStop Server	
Biblioteca	**Descripción**
libmqic.so	OSS sin hebras
libmqic_r.so	OSS multihebra
mqic	Guardian

Las aplicaciones de IBM WebSphere MQ nativas multihebra deben utilizar la característica PUT (Posix User Threads). No se da soporte a SPT (Standard Posix Threads) en este producto.

Creación de aplicaciones utilizando el compilador C de OSS

Esta sección contiene ejemplos de los mandatos que se utilizan para crear programas diseñados para OSS o Guardian cuando se utiliza el compilador de OSS.

`MQ_INSTALLATION_PATH` representa el directorio de alto nivel en el que está instalado IBM WebSphere MQ .

En el ejemplo siguiente, se crea una aplicación de OSS de cliente C sin hebras:

```
c89 -Wsystype=oss -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
```

En el ejemplo siguiente, se crea una aplicación de OSS de cliente C multihebra:

```
c89 -Wsystype=oss -D_PUT_MODEL_ -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic_r -lput
```

En el ejemplo siguiente, se crea una aplicación de cliente C de Guardian:

```
c89 -Wsystype=guardian -o /G/vol/subvol/amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
```

Creación de aplicaciones utilizando el compilador C de Guardian

Esta sección contiene ejemplos de los mandatos que se utilizan para crear programas diseñados para Guardian cuando se utiliza el compilador de Guardian.

MQ_INSTALLATION_PATH representa el volumen y subvolumen de Guardian en el que está instalado IBM WebSphere MQ .

En el ejemplo siguiente, se crea una aplicación de cliente C de Guardian:

```
CCOMP /in AMQSPUT0/ AMQSPUTC;&
       runnable,systype guardian,nolist,&
       ssv0 "$system.system",&
       ssv1 "MQINSTALLATION_SUBVOL",&
       ld(-LMQINSTALLATION_SUBVOL -lmqic)
```

Preparación de programas COBOL

Este tema contiene información a tener en cuenta al preparar programas C para el cliente IBM WebSphere MQ para HP Integrity NonStop Server. Contiene ejemplos de mandatos que se utilizan cuando se compilan aplicaciones utilizando el compilador de ECOBOL de OSS y utilizando el compilador de ECOBOL de Guardian.

Para crear un ejemplo COBOL a partir de código fuente, utilice el compilador ECOBOL.

La tabla siguiente indica las bibliotecas necesarias cuando está preparando programas COBOL en HP Integrity NonStop Server. Debe enlazar sus programas con la biblioteca adecuada que proporciona IBM WebSphere MQ.

Tabla 64. . Bibliotecas de enlace de HP Integrity NonStop Server

Biblioteca	**Descripción**
libmqic.so	OSS sin hebras
mqic	Guardian

Cuando ejecuta una aplicación en COBOL que se conecta a un gestor de colas, primero debe establecer la variable *SAVE-ENVIRONMENT* en 0N. Para definir la variable *SAVE-ENVIRONMENT* en 0N:

- Para OSS, escriba el mandato siguiente:

```
export SAVE-ENVIRONMENT=ON
```

- Para Guardian, escriba el mandato siguiente:

```
param SAVE-ENVIRONMENT ON
```

Si no establece la variable *SAVE-ENVIRONMENT* en 0N, cuando la aplicación se intenta conectar a un gestor de colas, falla con el código de razón de 2058 (080A) (RC2058): MQRC_Q_MGR_NAME_ERROR.

Creación de aplicaciones utilizando el compilador ECOBOL de OSS

Esta sección contiene ejemplos de los mandatos que se utilizan para crear programas diseñados para OSS o Guardian cuando se utiliza el compilador ECOBOL de OSS.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado IBM WebSphere MQ .

En el ejemplo siguiente se crea una aplicación de OSS de cliente COBOL:

```
   ecobol -Wsystype=oss
          -Wcobol="ansi;port"
          -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
          -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
```

```
        -LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
        -o amq0put0
        MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

El ejemplo siguiente compila una aplicación Guardian de cliente de COBOL:

```
ecobol -Wsystype=guardian
       -Wcobol="ansi;port;save all"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

Compilación de aplicaciones utilizando el compilador ECOBOL de Guardian

Esta sección contiene ejemplos de los mandatos que se utilizan para compilar programas que están destinados a Guardian cuando se utiliza el compilador ECOBOL de Guardian.

MQ_INSTALLATION_SUBVOL representa el volumen Guardian y el subvolumen en que se instala IBM WebSphere MQ.

El ejemplo siguiente compila una aplicación Guardian de cliente de COBOL:

```
ECOBOL /in MQSPUTL/ MQSPUT,MQINSTALLATION_SUBVOL.cmqcobol;
       call-shared;ansi;port;save all;nolist;runnable;
       consult MQINSTALLATION_SUBVOL.mqicb;
       eld(-LMQINSTALLATION_SUBVOL -lmqic)
```

Preparación de programas pTAL

Aprenda a crear programas pTAL para el cliente IBM WebSphere MQ en la plataforma HP Integrity NonStop Server .

Para crear un ejemplo pTAL a partir del código fuente, utilice el compilador EPTAL.

Nota:

- Las aplicaciones pTAL de IBM WebSphere MQ deben utilizar una rutina principal que esté escrita en el lenguaje C o COBOL.
- Las aplicaciones pTAL solo se pueden crear en Guardian.

En la tabla siguiente, se lista la biblioteca que se necesita cuando se preparan programas pTAL en HP Integrity NonStop Server. Debe enlazar sus programas con la biblioteca adecuada que proporciona IBM WebSphere MQ.

Tabla 65. . Biblioteca de enlaces de HP Integrity NonStop Server

Biblioteca	**Descripción**
mqic	Guardian

Creación de aplicaciones utilizando el compilador EPTAL de Guardian

Esta sección contiene ejemplos de los mandatos que se utilizan para crear programas que están destinados a Guardian cuando se utiliza el compilador EPTAL de Guardian.

MQINSTALLATION_SUBVOL representa el volumen y el subvolumen de Guardian en los que se ha instalado IBM WebSphere MQ.

Las aplicaciones pTAL de IBM WebSphere MQ deben utilizar una rutina principal que esté escrita en el lenguaje C o COBOL.

En el ejemplo siguiente, se crea una aplicación de Guardian de cliente pTAL:

```
        ASSIGN SSV0, $SYSTEM.SYSTEM
        ASSIGN SSV1, MQINSTALLATION_SUBVOL
```

```
EPTAL /in MQINSTALLATION_SUBVOL.MQSPUTT/ MQSPUTO;nolist

CCOMP /in MQINSTALLATION_SUBVOL.MQSPTMC/ MQSPUT;
       runnable,systype guardian,extensions,nolist,
       ssv0 "$system.system",
       ssv1 "MQINSTALLATION_SUBVOL",
       eld(MQSPUTO -LMQINSTALLATION_SUBVOL -lmqic)
```

Creación de la aplicación en HP-UX

Esta información describe las tareas adicionales y los cambios en las tareas estándar que debe realizar al crear aplicaciones WebSphere MQ para HP-UX para que se ejecuten bajo HP-UX.

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 WebSphere MQ para HP-UX varían con el lenguaje de programación en el que está escrito 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 WebSphere MQ para HP-UX para el idioma que está utilizando. Familiarícese con el contenido de estos archivos. Consulte “IBM WebSphere MQ archivos de definición de datos” en la página 82 para obtener una descripción completa.

En este tema se utiliza una barra inclinada invertida (\) para dividir mandatos largos en más de una línea. No especifique ese carácter; escriba cada mandato en una sola línea.

Preparación de programas C en HP-UX

Este tema contiene información a tener en cuenta al preparar programas C en HP-UX; con ejemplos para la plataforma IA64 (IPF).

MQ_INSTALLATION_PATH representa el directorio de alto nivel en que está instalado WebSphere MQ.

Trabaje en el entorno habitual. Se proporcionan programas C precompilados en el directorio *MQ_INSTALLATION_PATH*/samp/bin.

Para obtener información adicional sobre programación de aplicaciones de64 bits, consulte Estándares de codificación en plataformas de 64 bits.

Para utilizar SSL, los clientes MQI de WebSphere MQ en HP-UX deben compilarse utilizando hebras POSIX .

Algunos ejemplos a tener en cuenta:

- “Plataforma IA64 (IPF)” en la página 450
- “Enlazar bibliotecas” en la página 452

Plataforma IA64 (IPF)

Ejemplos de compilación de amqsput0, cliexit y srvexit en una plataforma IA64(IPF).

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación cliente en un entorno de 32 bits sin hebras:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación cliente en un entorno de 32 bits con hebras:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación cliente en un entorno de 64 bits sin hebras:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación cliente en un entorno de 64 bits con hebras:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación servidora en un entorno de 32 bits sin hebras:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación servidora en un entorno de 32 bits con hebras:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación servidora en un entorno de 64 bits sin hebras:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

En el ejemplo siguiente se compila el programa de ejemplo amqsput0 como una aplicación servidora en un entorno de 64 bits con hebras:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

En el ejemplo siguiente se compila la salida de cliente cliexit en un entorno de 32 bits sin hebras:

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

En el ejemplo siguiente se compila la salida de cliente cliexit en un entorno de 32 bits con hebras:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

En el ejemplo siguiente se compila la salida de cliente cliexit en un entorno de 64 bits sin hebras:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

En el ejemplo siguiente se compila la salida de cliente cliexit en un entorno de 64 bits con hebras:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
```

```
ld  -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

En el ejemplo siguiente se compila la salida de servidor srvexit en un entorno de 32 bits sin hebras:

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

En el ejemplo siguiente se compila la salida de servidor srvexit en un entorno de 32 bits con hebras:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

En el ejemplo siguiente se compila la salida de servidor srvexit en un entorno de 64 bits sin hebras:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c
-IMQ_INSTALLATION_PATHMQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

En el ejemplo siguiente se compila la salida de servidor srvexit en un entorno de 64 bits con hebras:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Enlazar bibliotecas

Debe enlazar los programas con la biblioteca adecuada proporcionada por WebSphere MQ.

En la tabla siguiente se muestra qué biblioteca utilizar en los distintos entornos

Plataforma de hardware	Entorno con hebras sin hebras	Tipo programa/salida	Archivo de biblioteca
IA64 (IPF)	Con hebras	Cliente y servidor en C	libmqm_r.so
IA64 (IPF)	Con hebras	Cliente en C	libmqic_r.so
IA64 (IPF)	Sin hebras	Cliente y servidor en C	libmqm.so
IA64 (IPF)	Sin hebras	Cliente en C	libmqic.so

Nota:

1. Si está escribiendo un servicio instalable (consulte Administración para obtener más información), tiene que enlazar con la biblioteca `libmqmzf.sl`.
2. Si está generando una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como, por ejemplo, IBM TXSeries Encinao BEA Tuxedo, debe enlazar con las bibliotecas `libmqmxa.sl` (o `libmqmxa64.sl` si el gestor de transacciones trata el tipo 'long' como de 64 bits) y `libmqz.sl` en una aplicación sin hebras y con las bibliotecas `libmqmxa_r.sl` (o `libmqmxa64_r.sl`) y `libmqz_r.sl` en una aplicación con hebras.
3. Debe enlazar las bibliotecas de WebSphere MQ antes que cualquier otra biblioteca de producto.

Preparación de programas COBOL en HP-UX

Información sobre cómo preparar programas COBOL en HP-UX, utilizando Micro Focus Server Express con WebSphere MQ en la plataforma IA64 (IPF) y ejecutando programas en el entorno de cliente MQI de WebSphere MQ .

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Notas a los usuarios

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. Los libros de copias 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
   ```

   en aplicaciones de 64 bits.

Compile los programas utilizando el compilador Micro Focus. Los archivos de copias que declaran las estructuras se encuentran en *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

Compilación de programas de 32 bits:

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Compilación de programas de 64 bits:

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

donde `amqsput` es un programa de ejemplo

Asegúrese de haber especificado los tamaños de pila de tiempo de ejecución adecuados; 16 KB es el mínimo recomendado.

Debe enlazar los programas con la biblioteca adecuada proporcionada por WebSphere MQ. En la tabla siguiente se muestra qué biblioteca utilizar en los distintos entornos

Plataforma de hardware	**Tipo programa/salida**	**Archivo de biblioteca**
IA64 (IPF)	Servidor para COBOL	libmqmcb.so
IA64 (IPF)	Cliente para COBOL	libmqicb.so

Plataforma de hardware	Tipo programa/salida	Archivo de biblioteca
IA64 (IPF)	Aplicaciones con hebras	libmqmcb_r.so

Utilización de Micro Focus Server Express con WebSphere MQ en la plataforma IA64 (IPF)

Consulte “Modelos de espacio de direcciones soportados por WebSphere MQ para HP-UX en IA64 (IPF)” en la página 455 para obtener detalles sobre cómo utilizar Micro Focus Server Express junto con WebSphere MQ en la plataforma HP/IPF.

Programas para ejecutar en el entorno de cliente MQI de WebSphere MQ

Si utiliza LU 6.2 para conectar su cliente MQI a un servidor, enlace su aplicación a libsna.a, que forma parte del producto SNAplusAPI. Utilice las opciones `-lV3` y `-lstr` en el mandato de compilación y enlace.

- La opción `-lV3` proporciona al programa acceso a la biblioteca de señalización AT & T (la SNAplusAPI utiliza señales AT & T)
- La opción `-lstr` enlaza su programa con el componente de corrientes

Preparación de programas CICS en HP-UX

Aprenda a crear programas de transacción CICS en HP-UX.

Para crear la transacción CICS de ejemplo, amqscic0.ccs, ejecute el mandato siguiente:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

Se proporciona un módulo de conmutador XA para permitirle enlazar CICS con WebSphere MQ:

Tabla 66. Código esencial para aplicaciones CICS (HP-UX)		
Descripción	**C (fuente)**	**C (ejec)**
Rutina de inicialización XA	amqzscix.c	amqzsc

Puede encontrar más información sobre el soporte de transacciones CICS en Administración.

Soporte de TXSeries CICS

WebSphere MQ en HP-UX da soporte a TXSeries CICS utilizando la interfaz XA. Asegúrese de que las aplicaciones CICS estén enlazadas a la versión con hebras de las bibliotecas de MQ .

Escriba programas WebSphere MQ que se carguen en la misma región CICS en C o COBOL. No puede realizar una combinación de llamadas MQI de C y COBOL en la misma región CICS . La mayoría de las llamadas MQI del segundo lenguaje utilizado fallan, y se emite un código de razón de `MQRC_HOBJ_ERROR`.

Transacción de ejemplo C CICS

El origen C de ejemplo para una transacción CICS WebSphere MQ lo proporciona AMQSCIC0.CCS. 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 el nombre de cola 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.

Preparación de programas CICS COBOL utilizando Micro Focus COBOL

MQ_INSTALLATION_PATH representa el directorio de alto nivel en que está instalado WebSphere MQ.

Para utilizar Micro Focus COBOL, siga estos pasos:

1. Añada el módulo de biblioteca de tiempo de ejecución COBOL WebSphere MQ a la biblioteca de tiempo de ejecución utilizando el mandato siguiente:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Nota: Con `cicsmkcobol`, WebSphere MQ no le permite realizar llamadas MQI en el lenguaje de programación C desde la 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 mover estas funciones, no incluya la biblioteca WebSphere MQ `libmqmcbrt.o` al crear 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 Micro Focus COBOL y habilita la biblioteca COBOL de tiempo de ejecución CICS para llamar a WebSphere MQ en sistemas UNIX and Linux .

   Nota: Ejecute `cicsmkcobol` solo cuando instale uno de los productos siguientes:

   - Versión o release nuevo de Micro Focus COBOL
   - Nueva versión o release de CICS para HP-UX
   - Versión o release nuevo de cualquier producto de base de datos soportado (solo para las transacciones COBOL)
   - Nueva versión o release de WebSphere 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
   ```

Modelos de espacio de direcciones soportados por WebSphere MQ para HP-UX en IA64 (IPF)

HP-UX proporciona varios modelos de espacio de direcciones que pueden ser explotados por aplicaciones WebSphere MQ .

HP-UX da soporte a dos modelos de espacio de direcciones:

- MGAS-Espacio de direcciones global en su mayoría (este es el valor predeterminado y lo utiliza WebSphere MQ)
- MPAS - Mostly Private Address Space

Las aplicaciones que se conectan a WebSphere MQ pueden utilizar los modelos de espacio de direcciones MGAS o MPAS. Las aplicaciones creadas utilizando el modelo MPAS que se conectan a WebSphere MQ utilizando memoria compartida pueden incurrir en un coste de rendimiento menor debido a la ineficiencia en la correlación de las páginas de memoria compartida utilizadas por WebSphere MQ en el espacio de direcciones virtual del programa MPAS.

Las aplicaciones COBOL creadas con Micro Focus Server Express utilizan el modelo MPAS de forma predeterminada.

Puede utilizar el programa **chatr** para comprobar y cambiar el modelo de direccionamiento que utiliza un programa.

Si encuentra problemas al conectarse a WebSphere MQ desde programas MPAS de 32 bits, considere la posibilidad de utilizar el modelo de direccionamiento MGAS o de crear la aplicación como una aplicación MPAS de 64 bits en lugar de una aplicación MPAS de 32 bits.

Puede encontrar más detalles sobre los modelos de espacio de direcciones MGAS y MPAS en la documentación de HP-UX .

Creación de la aplicación en Linux

Esta información describe las tareas adicionales y los cambios en las tareas estándar que debe realizar al crear WebSphere MQ para que se ejecuten las aplicaciones Linux .

Se da soporte a C y C++. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++.

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 WebSphere 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.

- Debe enlazar los programas con la biblioteca adecuada proporcionada por WebSphere MQ.

  En un entorno sin hebras, enlace con 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 con 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. Si escribe un servicio instalable (para obtener más información, consulte Administración), debe enlazar a la biblioteca `libmqmzf.so`.
  2. Si está generando una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como, por ejemplo, IBM TXSeries Encinao BEA Tuxedo, debe enlazar con las bibliotecas `libmqmxa.so` (o `libmqmxa64.so` si el gestor de transacciones trata el tipo 'long' como de 64 bits) y `libmqz.so` en una aplicación sin hebras y con las bibliotecas `libmqmxa_r.so` (o `libmqmxa64_r.so`) y `libmqz_r.so` en una aplicación con hebras.
  3. Debe enlazar las bibliotecas de WebSphere MQ antes que cualquier otra biblioteca de producto.

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 WebSphere MQ.

Aplicación de cliente en C de 31 bits sin hebras

```
gcc -m31  -o famqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicación de cliente en C de 31 bits con hebras

```
gcc -m31  -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicación de servidor en C de 31 bits sin hebras

```
gcc -m31  -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicación de servidor en C de 31 bits con hebras

```
gcc -m31  -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicación de cliente en C++ de 31 bits sin hebras

```
g++ -m31 -fsigned-char  -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicación de cliente en C++ de 31 bits con hebras

```
g++ -m31 -fsigned-char  -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Aplicación de servidor en C++ de 31 bits sin hebras

```
g++ -m31 -fsigned-char  -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicación de servidor en C++ de 31 bits con hebras

```
g++ -m31 -fsigned-char  -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Salida de cliente en C de 31 bits sin hebras

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

Salida de cliente en C de 31 bits con hebras

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

Salida de servidor en C de 31 bits sin hebras

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm
```

Salida de servidor en C de 31 bits con hebras

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32_r srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm_r -lpthread
```

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 WebSphere MQ.

Aplicación de cliente en C de 32 bits sin hebras

```
gcc -m32 -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Aplicación de cliente en C de 32 bits con hebras

```
gcc -m32 -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Aplicación de servidor en C de 32 bits sin hebras

```
gcc -m32 -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Aplicación de servidor en C de 32 bits con hebras

```
gcc -m32 -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Aplicación de cliente en C++ de 32 bits sin hebras

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicación de cliente en C++ de 32 bits con hebras

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Aplicación de servidor en C++ de 32 bits sin hebras

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicación de servidor en C++ de 32 bits con hebras

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Salida de cliente en C de 32 bits sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

Salida de cliente en C de 32 bits con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

Salida de servidor en C de 32 bits sin hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Salida de servidor en C de 32 bits con hebras

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
lmqm_r -lpthread
```

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 WebSphere MQ.

Aplicación cliente en C, 64 bits, sin hebras

```
gcc -m64  -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Aplicación cliente en C, 64 bits, con hebras

```
gcc -m64  -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Aplicación de servidor en C, 64 bits, sin hebras

```
gcc -m64  -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Aplicación de servidor en C, 64 bits, con hebras

```
gcc -m64  -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Aplicación cliente en C++, 64 bits, sin hebras

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación cliente en C++, 64 bits, con hebras

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

aplicación de servidor en C++, 64 bits, sin hebras

```
g++ -m64 -fsigned-char  -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicación de servidor en C++, 64 bits, con hebras

```
g++ -m64 -fsigned-char  -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Salida cliente en C , 64 bits, sin hebras

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Salida cliente en C , 64 bits, con hebras

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64_r cliexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Salida de servidor en C, 64 bits, sin hebras

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Salida de servidor en C, 64 bits, con hebras

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64_r srvexit.c
```

```
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Preparación de programas en COBOL en Linux

Obtenga información sobre la preparación de programas COBOL en Linux y la preparación de programas COBOL utilizando Micro Focus COBOL.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere 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 las plataformas de 64 bits, se instalan los libros de copia COBOL de 64 bits 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
   ```

   en 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)
libmqicb_r.so	Cliente para COBOL (aplicación con hebras)

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>
export LIB=MQ_INSTALLATION_PATHlib:$LIB
```

Para compilar un programa COBOL de 32 bits, cuando haya soporte, usando Micro Focus COBOL, especifique:

```
  $ 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, especifique:

```
$ 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.

Creación de la aplicación en Solaris

Esta información describe las tareas adicionales y los cambios en las tareas estándar que debe realizar al crear WebSphere MQ para que las aplicaciones Solaris se ejecuten en Solaris.

Están soportados los lenguajes de programación COBOL, C y C++. Para obtener información acerca de cómo preparar programas C++, consulte la sección Utilización de C++.

Además de codificar las llamadas MQI de su código fuente, debe añadir los archivos de inclusión adecuados. Familiarícese con el contenido de estos archivos. Consulte “IBM WebSphere MQ archivos de definición de datos” en la página 82 para obtener una descripción completa.

En este tema, se utiliza el carácter de barra invertida (\) para dividir los mandatos largos en más de una línea. No especifique este carácter, especifique cada mandato en una sola línea.

Preparación de programas C en Solaris

Los programas C precompilados se proporcionan en el directorio *MQ_INSTALLATION_PATH*/`samp/bin`.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Para obtener más información acerca de cómo programar aplicaciones de 64 bits, consulte la sección Estándares de codificación en las plataformas de 64 bits.

Si desea utilizar los programas en una máquina que sólo tiene instalado el cliente MQI de WebSphere MQ para Solaris, compile los programas para enlazarlos con la biblioteca de cliente (`-lmqic`).

Si utiliza el compilador no soportado `?usr?ucb?cc`, es posible que la aplicación se compile y enlace correctamente. No obstante, cuando ejecute la aplicación fallará cuando intente conectarse al gestor de colas.

Nota: Los clientes SSL y TLS Solaris x86 de 32 bits configurados para la operación compatible con FIPS 140-2 fallan cuando se ejecutan en sistemas Intel. Este error se produce porque el archivo de biblioteca de GSKit-Crypto Solaris x86 de 32 bits compatible con FIPS 140-2 no se carga en el conjunto de chips de Intel. En los sistemas afectados, el error AMQ9655 se notifica en el registro de errores de cliente. Para resolver este problema, inhabilite la compatibilidad con FIPS 140-2 o bien vuelva a compilar la aplicación cliente de 64 bits, porque el código de 64 bits no está afectado.

Enlazar bibliotecas

Debe enlazar con las bibliotecas de WebSphere MQ que sean adecuadas para el tipo de aplicación:

Archivos de biblioteca	Tipo programa/salida
libmqm.so	Servidor en C
libmqic.so & libmqm.so	Cliente en C

Nota:

1. Si está escribiendo un servicio instalable (para obtener más información, consulte Administración), enlace a la biblioteca `libmqmzf.so` .
2. Si está generando una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como IBM TXSeries Encinao BEA Tuxedo, debe enlazar con las bibliotecas `libmqmxa.so` (o `libmqmxa64.so` si el gestor de transacciones trata el tipo 'long' como de 64 bits) y `libmqz.so` .
3. Debe enlazar las bibliotecas de WebSphere MQ antes que cualquier otra biblioteca de producto.

Creación de aplicaciones en x86-64

Este tema contiene ejemplos de los comandos utilizados para compilar programas en diversos entornos de la plataforma x86-64.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Aplicación cliente en C, 32 bits

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplicación cliente en C, 64 bits

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

Aplicación de servidor en C, 32 bits

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplicación de servidor en C, 64 bits

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

Aplicación cliente en C++, 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

Aplicación cliente en C++, 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicación de servidor en C++, 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplicación de servidor en C++, 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Salida cliente en C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

Salida cliente en C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Salida de servidor en C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

Salida de servidor en C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Creación de aplicaciones en SPARC

Este tema contiene ejemplos de los comandos utilizados para compilar programas en diversos entornos de la plataforma SPARC.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Aplicación cliente en C, 32 bits

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplicación cliente en C, 64 bits

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Aplicación de servidor en C, 32 bits

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplicación de servidor en C, 64 bits

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Aplicación cliente en C++, 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

Aplicación cliente en C++, 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicación de servidor en C++, 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplicación de servidor en C++, 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Salida cliente en C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

Salida cliente en C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Salida de servidor en C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

Salida de servidor en C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Preparación de programas COBOL en Solaris

Información sobre la preparación de programas COBOL en Solaris.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere 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. Los libros de copias 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
```

en aplicaciones de 64 bits.

Compile los programas utilizando el compilador Micro Focus. Los archivos de copia que declaran las estructuras están en *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

Compilación de programas de 32 bits:

- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb`

  Servidor para COBOL
- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb`

  Cliente para COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r`

  Servidor de COBOL con hilos
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r`

  Cliente COBOL con hilos

Compilación de programas de 64 bits:

- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb`

  Servidor para COBOL
- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb`

  Cliente para COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r`

  Servidor de COBOL con hilos
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r`

  Cliente COBOL con hilos

donde *amqs0put0.cbl* es un programa de ejemplo.

Debe enlazar el programa con uno de los siguientes:

- libmqmcb.so

  Servidor para COBOL

- libmqicb.so

  Cliente para COBOL

Preparación de programas CICS en Solaris

Información sobre la preparación de programas CICS en Solaris.

Se proporciona un módulo de conmutador XA para permitirle enlazar CICS con WebSphere MQ:

Tabla 67. Código esencial para aplicaciones CICS (Solaris)		
Descripción	**C (fuente)**	**C (ejec)**
Rutina de inicialización XA	amqzscix.c	amqzsc - TXSeries para Solaris

Enlace siempre las transacciones con la biblioteca de WebSphere MQ de hebra segura libmqm.so.

Puede encontrar más información sobre el soporte de transacciones CICS en Administración.

Soporte de TXSeries CICS

WebSphere MQ para Solaris da soporte a TXSeries CICS utilizando la interfaz XA.

Escriba programas WebSphere MQ que se carguen en la misma región CICS en C o COBOL. No puede realizar una combinación de llamadas MQI de C y COBOL en la misma región 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 Micro Focus COBOL

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Para utilizar Micro Focus COBOL, siga estos pasos:

1. Añada el módulo de biblioteca de tiempo de ejecución COBOL WebSphere MQ a la biblioteca de tiempo de ejecución utilizando el mandato siguiente:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
   ```

   Nota: Con `cicsmkcobol`, WebSphere MQ no le permite realizar llamadas MQI en el lenguaje de programación C desde la aplicación COBOL.

   Si las aplicaciones existentes tienen llamadas de este tipo, mueva estas funciones de las aplicaciones COBOL a su propia biblioteca, por ejemplo, `myMQ.so`. Después de mover estas funciones, no incluya la biblioteca WebSphere MQ `libmqmcbrt.o` al crear 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 Micro Focus COBOL y habilita la biblioteca COBOL de tiempo de ejecución CICS para llamar a WebSphere MQ en sistemas UNIX and Linux .

   Nota: Ejecute `cicsmkcobol` solo cuando instale uno de los productos siguientes:

   - Versión o release nuevo de Micro Focus COBOL
   - Nueva versión o release de TXSeries para Solaris
   - Versión o release nuevo de cualquier producto de base de datos soportado (solo para las transacciones COBOL)
   - Nueva versión o release de WebSphere 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 CICS C

Cree programas CICS C utilizando los recursos CICS estándar:

1. Exporte ***una*** de las variable de entorno siguientes:
   - LDFLAGS = "-L*MQ_INSTALLATION_PATH*ilib -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 de ejemplo C CICS

El origen C de ejemplo para una transacción CICS WebSphere MQ lo proporciona AMQSCIC0.CCS. 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.

Creación de la aplicación en sistemas Windows

Las publicaciones de los sistemas Windows describen cómo crear aplicaciones ejecutables a partir de los programas que escribe.

En este tema se describen las tareas adicionales y los cambios en las tareas estándar que debe realizar al crear aplicaciones WebSphere MQ para Windows para que se ejecuten en sistemas Windows . Están soportados los lenguajes de programación ActiveX, C, C++, COBOL y Visual Basic. Para obtener información acerca de cómo preparar los programas ActiveX, consulte Utilización de la interfaz del Modelo de objetos componentes (WebSphere MQ Automation Classes for ActiveX). 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 WebSphere MQ para Windows varían con el lenguaje de programación en el que está escrito 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 WebSphere MQ para Windows para el idioma que está utilizando. Familiarícese con el contenido de estos archivos. Consulte “IBM WebSphere MQ archivos de definición de datos” en la página 82 para obtener una descripción completa.

Creación de aplicaciones de 64 bits en Windows

En IBM WebSphere MQ for Windows Version 7.5 se admiten aplicaciones de 32 bits y de 64 bits. Los archivos ejecutables y de biblioteca de IBM WebSphere 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 WebSphere MQ se proporcionan en las ubicaciones siguientes:

Tabla 68. Ubicación de las bibliotecas de IBM WebSphere 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 WebSphere 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.

Preparación de programas C en Windows

Trabaje en su entorno típico de Windows ; WebSphere MQ for Windows no requiere nada especial.

Para obtener más información sobre la programación de aplicaciones de 64 bits, consulte Estándares de codificación en plataformas de 64 bits.

- Enlace los programas con las bibliotecas adecuadas proporcionadas por WebSphere 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
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 WebSphere MQ.

El mandato siguiente proporciona un ejemplo de compilación del programa de ejemplo amqsget0 (utilizando el compilador 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 para obtener más información), necesitará enlazar con la biblioteca mqmzf.lib.
- Si está generando una aplicación para la coordinación externa mediante un gestor de transacciones compatible con XA como IBM TXSeries Encinao BEA Tuxedo, debe enlazar con la biblioteca mqmxa.lib o mqmxa.lib .
- Si está escribiendo una salida CICS , enlace a la biblioteca mqmcics4.lib .
- Debe enlazar las bibliotecas de WebSphere MQ antes que cualquier otra biblioteca de 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 WebSphere MQ para Windows a WebSphere MQ en sistemas UNIX and Linux , donde es necesario utilizar minúsculas.

Preparación de programas CICS y Transaction Server

El origen C de ejemplo para una transacción CICS WebSphere MQ lo proporciona AMQSCIC0.CCS. Puede compilarlo utilizando los recursos estándar 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=-IMQ_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 WebSphere MQ.

Esto se describe en la publicación *Transaction Server for Windows NT Application Programming Guide (CICS) V4*.

Puede encontrar más información sobre el soporte de transacciones CICS en Administración.

Preparación de programas en 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 copias de COBOL de 32 bits se instalan en el directorio siguiente: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. Los libros de copias 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
   ```

en aplicaciones de 64 bits.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.

Para preparar programas COBOL en sistemas Windows, enlace el programa a una de las bibliotecas siguientes proporcionadas por IBM WebSphere MQ:

Archivo de biblioteca	Tipo de salida o programa
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb	Servidor de 32 bits para IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb	Servidor de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb	Cliente de 32 bits para IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb	Cliente de 32 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb	Servidor de 64 bits para IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb	Servidor de 64 bits para Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb	Cliente de 64 bits para IBM 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 WebSphere MQ.

Se puede usar el compilador IBM COBOL Set o el compilador Micro Focus COBOL, dependiendo del programa:

- Los programas que comienzan por `amqi` son adecuados para el compilador IBM COBOL Set,
- 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.

IBM y Micro Focus COBOL

Vuelva a enlazar cualquier programa IBM WebSphere MQ Micro Focus COBOL de 32 bits existente utilizando `mqmcb.lib` o `mqiccb.lib`, en lugar de las bibliotecas `mqmcbb` y `mqiccbb`.

Para compilar, por ejemplo, el programa de ejemplo `amq0put0`, utilizando IBM VisualAge COBOL:

1. Establezca la variable de entorno SYSLIB para incluir la vía de acceso a los libros de copias COBOL de IBM WebSphere MQ VisualAge (especifique el código siguiente en una línea):

   ```
   set SYSLIB=MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook\VAcobol;%SYSLIB%
   ```

2. Para su uso en el servidor de IBM WebSphere MQ :

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqmcbb.lib"
   ```

3. Para su uso en el cliente de IBM WebSphere MQ :

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqiccbb.lib"
   ```

   Nota: Aunque debe utilizar la opción de compilador `CALLINT(SYSTEM)`, este es el valor predeterminado para `cob2`.

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 WebSphere 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 de IBM WebSphere MQ :

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - O bien, enlace el archivo de objeto para su uso en el cliente de IBM WebSphere 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
```

Preparación de programas de Visual Basic en Windows

Utilice esta información cuando considere la posibilidad de utilizar programas de Visual Basic en Windows.

Nota: No se proporcionan las versiones de 64 bits de los archivos del módulo 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 PCF
CMQXB.BAS	Soporte de salidas de canal
CMQPSB.BAS	Publicación/suscripción

Consulte “Codificación en Visual Basic” en la página 89 para obtener información sobre cómo utilizar 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 cliente WebSphere MQ , antes de compilar o ejecutar el proyecto, estableciendo la variable de compilación condicional *MqType*. Establezca *MqType* en un proyecto de Visual Basic en 1 para un servidor o 2 para un cliente como se indica 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
   ```

Salida de seguridad SSPI

WebSphere MQ para Windows proporciona una salida de seguridad para el cliente MQI de WebSphere MQ y el servidor de WebSphere MQ . Este es un programa de salida de canal que proporciona autenticación para canales WebSphere MQ utilizando la interfaz de programación de servicios de seguridad (SSPI). El 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 el apartado "Utilización de la salida de seguridad SSPI en sistemas Windows" en la página 173.

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 69. 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 acceso de WebSphere MQ y principales de Windows" en la página 475). 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 acceso de WebSphere MQ y principales de Windows

El control de acceso que proporciona WebSphere MQ se basa en el usuario y el grupo. La autenticación que proporciona Windows se basa en principales, como el usuario y el nombre servicePrincipal(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 asociado con el principal de Windows a WebSphere MQ para el control de acceso.

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 con el 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 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 nombre servicePrincipalque se forma añadiendo el prefijo siguiente al nombre QueueManager.

```
ibmMQSeries/
```

Utilización de servicios ligeros de protocolo de acceso a directorios con WebSphere MQ para Windows

En este tema se explica qué es un servicio de directorio y el papel que desempeña un protocolo de acceso a directorios (DAP). También explica cómo las aplicaciones de WebSphere MQ pueden utilizar un directorio LDAP (Lightweight Directory Access Protocol) utilizando un programa de ejemplo como guía.

Nota: El programa de ejemplo está diseñado para alguien que ya está familiarizado con LDAP.

Los temas siguientes proporcionan más información sobre los servicios de directorio, LDAP y la utilización de LDAP con WebSphere MQ.

- “Servicio de directorio” en la página 475
- “Lightweight Directory Access Protocol (LDAP)” en la página 476
- “Utilización de LDAP con WebSphere MQ” en la página 476

Servicio de directorio

Un directorio es un repositorio de información sobre objetos, que se organiza de tal manera que es fácil encontrar la información sobre un objeto específico.

Un ejemplo común es un directorio telefónico, donde se almacena información (dirección y número de teléfono) sobre personas y empresas. Otro ejemplo es una libreta de direcciones para un sistema de

correo electrónico, donde las direcciones de correo electrónico, y opcionalmente otra información como los números de teléfono, se almacenan para las personas.

En sistemas informáticos, los directorios pueden almacenar información sobre recursos informáticos, como impresoras o discos compartidos. Por ejemplo, puede utilizar un directorio para averiguar dónde se encuentra la impresora de color más cercana. En una aplicación WebSphere MQ se puede utilizar un directorio para proporcionar la asociación entre un servicio de aplicaciones (como el proceso de cuentas por cobrar) y la cola que se utilizará para los mensajes que requieran dicho servicio (posiblemente identificado mediante el nombre de cola y su nombre de gestor de colas de host).

Los directorios se implementan como sistemas cliente-servidor, donde el servidor de directorios contiene toda la información y responde a las peticiones de los clientes. Los clientes pueden ser programas de interfaz de usuario, que proporcionan la información directamente al usuario, o programas de aplicación que necesitan localizar recursos para completar su trabajo. Un servicio de directorio consta del servidor de directorios, los programas administrativos y las bibliotecas de cliente y los programas necesarios para configurar, actualizar y leer el directorio.

Lightweight Directory Access Protocol (LDAP)

Existen muchos servicios de directorio como, por ejemplo, Novell Directory Services, DCE Cell Directory Service, Banyan StreetTalk, Windows Directory Services, X.500y los servicios de libreta de direcciones asociados a los productos de correo electrónico. X.500 fue propuesto como estándar para los servicios de directorio global por la Organización Internacional de Estándares (ISO). Requiere una pila de protocolos OSI para sus comunicaciones, y en gran parte debido a esto, su uso se ha restringido a grandes organizaciones e instituciones académicas. Un servidor de directorios X.500 se comunica con sus clientes utilizando el protocolo de acceso a directorios (DAP).

LDAP (Lightweight Directory Access Protocol) se ha creado como una versión simplificada de DAP. Es más fácil de implementar, omite algunas de las características menos utilizadas de DAP y se ejecuta a través de TCP/IP. Como resultado de estos cambios, se está adoptando rápidamente como el protocolo de acceso a directorios para la mayoría de los fines, sustituyendo la multitud de protocolos propietarios utilizados anteriormente. Los clientes LDAP todavía pueden acceder a un servidor X.500 a través de una pasarela (X.500 todavía requiere la pila de protocolos OSI), o cada vez más implementaciones X.500 suelen incluir soporte nativo para LDAP, así como acceso DAP.

Los directorios LDAP se pueden distribuir y pueden utilizar la réplica para habilitar un acceso eficiente a su contenido.

Para obtener una descripción más completa de LDAP, consulte *Descripción de LDAP*, una publicación de IBM Redbooks .

Utilización de LDAP con WebSphere MQ

En configuraciones de WebSphere MQ , la información que define las colas de mensajes y de transmisión se almacena localmente. Esto significa que en una red de WebSphere MQ se distribuyen las distintas definiciones, sin que haya ningún directorio central de esta información disponible para su examen. La mensajería remota entre aplicaciones de WebSphere MQ se consigue normalmente mediante el uso de definiciones locales de colas remotas. La aplicación emite primero una llamada MQOPEN utilizando el nombre especificado en la definición local de la cola remota. Para colocar el mensaje en la cola remota, la aplicación emite MQPUT, especificando el descriptor de contexto devuelto por la llamada MQOPEN. La definición de cola remota proporciona el nombre de la cola de destino, el gestor de colas de destino y, opcionalmente, una cola de transmisión. En esta técnica, la aplicación tiene que conocer en tiempo de ejecución el nombre especificado en la definición de cola local.

Una variación en la anterior evita el uso de definiciones locales de colas remotas. La aplicación puede especificar el nombre de cola de destino completo, que incluye el nombre del gestor de colas remoto como parte de MQOPEN. Por lo tanto, la aplicación tiene que conocer estos dos nombres en tiempo de ejecución. El gestor de colas local debe estar configurado correctamente con la definición de cola local y con una cola de transmisión con el nombre adecuado (o predeterminado) y un canal asociado que se entrega al destino.

En el caso en el que los gestores de colas de origen y de destino estén definidos como miembros del mismo clúster, los aspectos de canal y cola de transmisión de los dos escenarios anteriores se pueden ignorar. Si la cola de transmisión de destino es una cola de clúster, tampoco es necesaria una definición local de una cola remota. Sin embargo, de forma similar a los casos anteriores descritos, la aplicación todavía debe conocer el nombre de la cola de destino.

Se puede utilizar un servicio de directorio para eliminar esta dependencia de aplicación en los nombres de cola (o la combinación de nombres de cola y de gestor de colas). La correlación entre los criterios de aplicación y los nombres de objeto de WebSphere MQ se puede guardar en un directorio y actualizarse dinámicamente, e independientemente de las aplicaciones. En tiempo de ejecución, la aplicación WebSphere MQ que desea enviar un mensaje primero consulta el directorio utilizando criterios basados en la aplicación, por ejemplo, donde: service_name = "accounts cobrar", recupera los nombres de objeto WebSphere MQ relevantes y, a continuación, utiliza estos valores devueltos en la llamada MQOPEN.

Otro ejemplo del uso de un directorio es para una empresa que tiene muchos depósitos u oficinas pequeños, WebSphere MQ Los clientes MQI se pueden utilizar para enviar mensajes a los servidores WebSphere MQ ubicados en las oficinas más grandes. Los clientes necesitan saber el nombre de la máquina host, el canal MQI y el nombre de cola para cada servidor al que envían mensajes. En ocasiones, puede ser necesario mover un servidor WebSphere MQ a otra máquina; cada cliente que se comunique con el servidor deberá conocer el cambio. Se puede utilizar un servicio de directorio LDAP para almacenar los nombres de las máquinas host (y los nombres de canal y cola) y los programas cliente pueden recuperar la información del directorio siempre que deseen enviar un mensaje a un servidor. En este caso, sólo es necesario actualizar el directorio si un nombre de host (o nombre de canal o cola) ha cambiado.

Se pueden almacenar varios destinos para un mensaje de aplicación en un directorio, en el que el elegido depende de consideraciones de disponibilidad o de compartimiento de carga.

WebSphere MQ también puede utilizar un directorio LDAP para almacenar información de autenticación para utilizarla con SSL (Secure Sockets Layer). WebSphere MQ classes for Java también puede almacenar información en un directorio LDAP.

programa de ejemplo LDAP

El programa de ejemplo está diseñado para alguien que esté familiarizado con LDAP y probablemente ya lo utilice. Está pensado para mostrar cómo las aplicaciones de WebSphere MQ pueden utilizar un directorio LDAP.

Creación del programa de ejemplo

Este programa se ha creado y probado sólo en Windows utilizando TCP/IP. Además de las consideraciones generales mencionadas en “Preparación de programas C en Windows” en la página 469, tenga en cuenta los puntos siguientes:

- Este programa está diseñado para ejecutarse como un programa cliente, por lo que debe enlazarse con MQIC.LIB .
- Además de los archivos y bibliotecas de cabecera de WebSphere MQ , este programa se debe crear utilizando archivos y bibliotecas de cabecera de cliente LDAP.

  Por ejemplo, utilizando el cliente IBM eNetwork , enlace el programa con LIBLDAPSTATICE.LIB y LIBLBERSTATICSSL.LIB .

configuración del directorio

Antes de que se pueda ejecutar el programa de ejemplo, se debe configurar un servidor de directorios LDAP con datos de ejemplo.

El archivo MQuser.ldif, en el directorio `tools\c\samples` , contiene algunos datos de ejemplo en LDIF (formato de intercambio de datos LDAP). Puede editar este archivo para que se ajuste a sus necesidades. Contiene datos para una empresa ficticia llamada MQuser que tiene un Departamento de Transporte que consta de tres oficinas. Cada una de estas oficinas tiene una máquina que ejecuta un servidor WebSphere MQ .

Como mínimo, debe editar las tres líneas que contienen los nombres de host de las máquinas que ejecutan los servidores WebSphere MQ : líneas 18, 27 y 36:

```
host: LondonHost
  ...
host: SydneyHost
  ...
host: WashingtonHost
```

Debe cambiar LondonHost, SydneyHosty WashingtonHost por los nombres de tres de las máquinas que ejecutan servidores WebSphere MQ . También puede cambiar los nombres de canal y cola si lo desea (el ejemplo utiliza nombres de los valores predeterminados del sistema). Es posible que también desee aumentar o disminuir el número de oficinas en los datos de ejemplo.

Configuración del servidor de directorios de IBM Tivoli

Consulte la publicación IBM Tivoli Directory Server (ITDS) Administrator's Guide para obtener información sobre la instalación del directorio. En el tema Installing and Configuring Server, consulte las secciones Installing Server y Basic Server Configuration. Si es necesario, lea el tema Administrator Interface para familiarizarse con el funcionamiento de la interfaz.

En el tema Configuring - How Do I, siga las instrucciones para iniciar el administrador y, a continuación, trabaje en la sección Configure Database y cree una base de datos predeterminada. Omita la sección Configure replica y, utilizando la sección Work with Suffixes, añada un sufijo o=MQuser.

Antes de añadir entradas a la base de datos, debe ampliar el esquema de directorio añadiendo algunas definiciones de atributo y una definición de clase de objeto. Esto se describe en la publicación IBM Tivoli Directory Server Guía del administrador en el capítulo Reference Information bajo la sección Directory Schema. Se incluyen dos archivos de ejemplo para ayudarle con esto. El archivo mq.at.conf incluye las definiciones de atributo que debe añadir al archivo ?etc?slapd.at.conf. Para ello, incluya el archivo de ejemplo editando slapd.at.conf y añadiendo una línea:

```
include <pathname>/mq.at.conf
```

De forma alternativa, puede editar el archivo slapd.at.conf y añadir el contenido del archivo de ejemplo directamente al mismo, es decir, añadir las líneas:

```
# MQ attribute definitions
attribute mqChannel           ces   mqChannel        1000   normal
attribute mqQueueManager      ces   mqQueueManager   1000   normal
attribute mqQueue             ces   mqQueue          1000   normal
attribute mqPort              cis   mqPort          64     normal
```

De forma similar para la definición de objectclass, puede incluir el archivo de ejemplo editando etc? slapd.oc.conf y añadir la línea:

```
include <pathname>/mq.oc.conf
```

o puede añadir el contenido del archivo de ejemplo directamente a slapd.oc.conf, , es decir, añadir las líneas:

```
# MQ object classdefinition
objectclass mqApplication
      requires
              objectClass,
              cn,
              host,
              mqChannel,
              mqQueue
      allows
              mqQueueManager,
```

```
              mqPort,
              description,
              l,
              ou,
              seeAlso
```

Ahora puede iniciar el servidor de directorios (Administración, Servidor, Inicio) y añadirle las entradas de ejemplo. Para añadir las entradas de ejemplo, vaya a la página Administración, Añadir entradas del administrador, escriba el nombre completo de la vía de acceso del archivo de ejemplo `MQuser.ldif` y pulse Enviar.

El servidor de directorios está ahora en ejecución y cargado con datos adecuados para ejecutar el programa de ejemplo.

Configuración del servidor de directorios Netscape

Utilizando la página Administración de Netscape Server, pulse **Crear nuevo Netscape Directory Server**.

Ahora se le debería presentar un formulario que contenga información de configuración. Cambie el sufijo de directorio a **o = MQuser** y añada una contraseña para el usuario sin restricciones. También puede cambiar cualquier otra información que se adapte a su instalación. Pulse **Aceptar**y el directorio se debe crear correctamente. Pulse **Volver a Administración del servidor** e inicie el servidor de directorios. Pulse el nombre del directorio para iniciar el servidor de administración de Directory Server para el nuevo directorio.

Antes de añadir entradas a la base de datos, amplíe el esquema de directorio añadiendo algunas definiciones de atributo y una definición de clase de objeto. Pulse el separador **Esquema** de la página Servidor de directorios. Ahora se le presenta un formulario que le permite añadir nuevos atributos. Añada los atributos siguientes (deje el OID de atributo en blanco para todos ellos):

```
Attribute Name              Syntax
--------------              ------
mqChannel                   Case Exact String
mqQueueManager              Case Exact String
mqQueue                     Case Exact String
mqPort                      Integer
```

Añada una nueva objectClass pulsando **Crear ObjectClass** en el panel lateral. Especifique **mqApplication** como ObjectClass Nombre, seleccione **applicationProcess** como padre ObjectClass y deje **OID de ObjectClass** en blanco. Ahora añada algunos atributos a objectClass. Seleccione **host**, **mqChannel**y **mqQueue** como atributos necesarios y seleccione **mqQueueManager** y **mqPort** como atributos permitidos. Pulse el botón **Crear clase de objeto ObjectClass** para crear la clase de objeto objectClass.

Para añadir los datos de ejemplo, pulse el separador **Gestión de bases de datos** y seleccione **Añadir entradas** en el panel lateral. Especifique el nombre de vía de acceso del archivo de datos de ejemplo `<pathname>\MQuser.ldif`, especifique la contraseña y pulse **Aceptar**.

El programa de ejemplo se ejecuta como un usuario no autorizado y, de forma predeterminada, Netscape Directory no permite a los usuarios no autorizados buscar en el directorio. Cámbielo pulsando el separador **Control de acceso** . Especifique la contraseña para el usuario sin restricciones y pulse **Aceptar** para cargar en las entradas de control de acceso para el directorio. Estos deben estar vacíos actualmente. Pulse el botón **Nuevo ACI** para crear una nueva entrada de control de acceso. En el recuadro de entrada que aparece, pulse **Denegar** (que está subrayado) y, en el recuadro de diálogo resultante, cámbielo a **Permitir**. Añada un nombre, por ejemplo, **MQuser-access**, y pulse **elegir un sufijo** para seleccionar **o = MQuser**. Especifique **o = MQuser** como destino, especifique la contraseña para el usuario sin restricciones y pulse **Enviar**.

El servidor de directorios está ahora en ejecución y cargado con datos adecuados para ejecutar el programa de ejemplo.

Ejecución del programa de ejemplo

Ahora debería tener un servidor de directorio LDAP en ejecución y rellenado con los datos de ejemplo. Los datos especifican tres máquinas host, todas las cuales deben ejecutar servidores WebSphere MQ .

Asegúrese de que el gestor de colas predeterminado se esté ejecutando en cada máquina (a menos que haya cambiado los datos de ejemplo para especificar un gestor de colas diferente).

Además, inicie el programa de escucha de WebSphere MQ en cada máquina; el ejemplo utiliza TCP/IP con el número de puerto predeterminado de WebSphere MQ , para que pueda iniciar el escucha con el mandato:

```
runmqlsr -t tcp
```

Para probar el ejemplo, es posible que también desee ejecutar un programa para leer los mensajes que llegan a cada servidor WebSphere MQ , por ejemplo, puede utilizar el programa de ejemplo amqstrg:

```
amqstrg SYSTEM.DEFAULT.LOCAL.QUEUE
```

El programa de ejemplo utiliza tres variables de entorno, una necesaria y dos opcionales. La variable necesaria es LDAP_BASEDN, que especifica el nombre distinguido base para la búsqueda de directorio. Para trabajar con los datos de ejemplo, establézcalo en `ou=Transport, o=MQuser`, por ejemplo, en un indicador de mandatos en sistemas Windows escriba:

```
set LDAP_BASEDN=ou=Transport, o=MQuser
```

Las variables opcionales son LDAP_HOST y LDAP_VERSION. La variable LDAP_HOST especifica el nombre del host donde se ejecuta el servidor LDAP; si no se especifica, el valor predeterminado es el host local. La variable LDAP_VERSION especifica la versión del protocolo LDAP que se va a utilizar y puede ser 2 o 3. La mayoría de los servidores LDAP ahora dan soporte a la versión 3 del protocolo; todos ellos dan soporte a la versión 2 anterior. Este ejemplo funciona igual de bien con cualquiera de las versiones del protocolo y, si no se especifica, toma el valor predeterminado de la versión 2.

Ahora puede ejecutar el ejemplo escribiendo el nombre de programa seguido del nombre de la aplicación WebSphere MQ a la que desea enviar mensajes, en el caso de los datos de ejemplo, los nombres de aplicación son Londres, Sídney y Washington. Por ejemplo, para enviar mensajes a la aplicación Londres:

```
amqsldpc London
```

Si el programa no se puede conectar con el servidor WebSphere MQ , aparecerá un mensaje de error adecuado. Si se conecta correctamente, puede empezar a escribir mensajes, cada línea que escriba (terminada por < return> o < enter>) se envía como un mensaje aparte, una línea vacía finaliza el programa.

Diseño de programa

El programa tiene dos partes distintas: la primera parte utiliza las variables de entorno y el valor de línea de mandatos para consultar un servidor de directorios LDAP; la segunda parte establece la conexión WebSphere MQ utilizando la información devuelta del directorio y envía los mensajes.

Las llamadas LDAP utilizadas en la primera parte del programa difieren ligeramente en función de si se está utilizando LDAP versión 2 o 3, y se describen en detalle en la documentación que viene con las bibliotecas de cliente LDAP. Esta sección ofrece una breve descripción.

La primera parte del programa comprueba que se ha llamado correctamente y lee las variables de entorno. A continuación, establece una conexión con el servidor de directorios LDAP en el host especificado:

```
if (ldapVersion == LDAP_VERSION3)
{
  if ((ld = ldap_init(ldapHost, LDAP_PORT)) == NULL)
     ...
}
else
{
  if ((ld = ldap_open(ldapHost, LDAP_PORT)) == NULL )
```

```
    ...
}
```

Cuando se ha establecido una conexión, el programa establece algunas opciones en el servidor con la llamada "ldap_set_option" y, a continuación, se autentica en el servidor enlazándolo:

```
if (ldapVersion == LDAP_VERSION3)
{
  if (ldap_simple_bind_s(ld, bindDN, password) != LDAP_SUCCESS)
    ...
}
else
{
  if (ldap_bind_s(ld, bindDN, password, LDAP_AUTH_SIMPLE) !=
      LDAP_SUCCESS)
    ...
}
```

En el programa de ejemplo `bindDN` y `password` se establecen en NULL, lo que significa que el programa se autentica como un usuario anónimo, es decir, no tiene ningún derecho de acceso especial y sólo puede acceder a la información disponible públicamente. En la práctica, la mayoría de las organizaciones restringen el acceso a la información que almacenan en directorios para que sólo los usuarios autorizados puedan acceder a ella.

El primer parámetro de la llamada de enlace `ld` es un descriptor de contexto que se utiliza para identificar esta sesión LDAP concreta en el resto del programa. Después de la autenticación, el programa busca en el directorio las entradas que coinciden con el nombre de la aplicación:

```
rc = ldap_search_s(ld,                 /* LDAP Handle             */
                   baseDN,             /* base distinguished name */
                   LDAP_SCOPE_ONELEVEL, /* one-level search        */
                   filterPattern,      /* filter search pattern   */
                   attrs,              /* attributes required     */
                   FALSE,              /* NOT attributes only     */
                   &ldapResult);       /* search result           */
```

Se trata de una simple llamada síncrona al servidor que devuelve los resultados directamente. Existen otros tipos de búsqueda que son más adecuados para consultas complejas o cuando se espera un gran número de resultados. El primer parámetro de la búsqueda es el descriptor de contexto `ld` que identifica la sesión. El segundo parámetro es el nombre distinguido base, que especifica dónde empieza la búsqueda en el directorio, y el tercer parámetro es el ámbito de la búsqueda, es decir, qué entradas relativas al punto de partida se buscan. Estos dos parámetros juntos definen qué entradas del directorio se buscan. El siguiente parámetro, `filterPattern` especifica lo que estamos buscando. El parámetro `attrs` lista los atributos que queremos recuperar del objeto cuando lo hayamos encontrado. El siguiente atributo indica si queremos sólo los atributos o sus valores también; si se establece en FALSE significa que queremos los valores de atributo. El parámetro final se utiliza para devolver el resultado.

El resultado podría contener muchas entradas de directorio, cada una con los atributos especificados y sus valores. Tenemos que extraer los valores que queremos del resultado. En este programa de ejemplo sólo esperamos que se encuentre una entrada, por lo que sólo miramos la primera entrada del resultado:

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

Esta llamada devuelve un descriptor de contexto que representa la primera entrada y configuramos un bucle for para extraer todos los atributos de la entrada:

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);
     attribute != NULL;
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))
{
```

Para cada uno de estos atributos, extraemos los valores asociados a él. De nuevo sólo esperamos un valor por atributo, por lo que sólo utilizamos el primer valor; determinamos qué atributo tenemos y almacenamos el valor en la variable de programa apropiada:

```
values = ldap_get_values(ld, ldapEntry, attribute);
if (values != NULL && values[0] != NULL)
{
  if (stricmp(attribute, MQ_HOST_ATTR) == 0)
  {
    mqHost = strdup(values[0]);
    ...
```

Finalmente, ordenamos liberando memoria (ldap_value_free, ldap_memfree, ldap_msgfree) y cerramos la sesión *desenlazando* del servidor:

```
ldap_unbind(ld);
```

Comprobamos que hemos encontrado todos los valores de WebSphere MQ que necesitamos del directorio y, si es así, llamamos a sendMessages() para conectar con el servidor de WebSphere MQ y enviar los mensajes de WebSphere MQ .

La segunda parte del programa de ejemplo es la rutina sendMessages() que contiene todas las llamadas de WebSphere MQ . Esto se modela en el programa de ejemplo amqsput0 , las diferencias son que los parámetros del programa se han ampliado y se utiliza MQCONNX en lugar de la llamada MQCONN.

Desarrollo de aplicaciones para IBM WebSphere 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 IBM WebSphere MQ Telemetry utilizando patrones de diseño, ejemplos preparados, programas de ejemplo, conceptos de programación e información de referencia. Utilice el daemon IBM WebSphere MQ Telemetry para dispositivos para simplificar la conexión de muchos dispositivos pequeños a IBM WebSphere MQ.

Conceptos relacionados

WebSphere MQ Telemetry

Conceptos y escenarios de telemetría para supervisar y controlar

Tareas relacionadas

Instalación de WebSphere MQ Telemetry

Administración de WebSphere MQ Telemetry

Resolución de problemas para WebSphere MQ Telemetry

Referencia relacionada

Referencia de WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry programas de ejemplo

Se proporcionan scripts de ejemplo para mostrar el uso básico de la aplicación cliente MQ Telemetry Transport v3 . Utilice los scripts para publicar un mensaje y suscribirse a un tema.

Antes de empezar

Inicie el servicio de telemetría (MQXR) para ejecutar los programas de ejemplo.

El ID de usuario debe ser miembro del grupo de usuarios mqm.

Ejecute primero el script SampleMQM , seguido del script MQTTV3Sample para realizar una publicación y suscripción. Ejecute el script de ejemplo CleanupMQM para suprimir el gestor de colas creado por el script SampleMQM .

A medida que el script SampleMQM crea y utiliza un gestor de colas denominado QM1, no se ejecuta sin modificar en un sistema con un gestor de colas QM1 . Cualquier cambio que se efectúe puede tener implicaciones en la configuración del gestor de colas existente.

Acerca de esta tarea

- La aplicación SampleMQM crea e inicia un gestor de colas habilitado para telemetría denominado QM1. El script también configura una cola de transmisión predeterminada para QM1, y crea e inicia un canal predeterminado a la escucha en el puerto 1883. Este canal no realiza ninguna autenticación de clientes conectados a él. El canal tiene el atributo de identificador de usuario de agente de canal de mensajes (MCAUSER), establecido en 'guest' en sistemas Windows o 'nobody' en sistemas Linux . Los clientes conectados al canal se tratan como el usuario 'guest' o el usuario 'nobody', dependiendo del sistema en el que se esté ejecutando. El script autoriza a 'guest' en sistemas Windows y a 'nobody' en sistemas Linux a poder publicar y suscribirse a cualquier tema en QM1
- La aplicación MQTTV3Sample se encuentra en la ubicación siguiente;
  - En Windows *MQ_INSTALLATION_PATH*\mqxr\samples

    donde *VÍA_INSTALACIÓN_MQ* es la ubicación en la que IBM WebSphere MQ está instalado.
  - En Linux *MQ_INSTALLATION_PATH*/mqxr/samples

  La aplicación MQTTV3Sample funciona como publicador, enviando un único mensaje a un tema del servidor. También actúa como suscriptor, escuchando los mensajes del servidor.
- El script de ejemplo CleanupMQM finaliza y suprime QM1 creado por el script SampleMQM . Utilice el script de ejemplo CleanupMQM si desea volver a ejecutar el script SampleMQM o eliminar QM1.

Procedimiento

1. Escriba el mandato siguiente en una línea de mandatos para ejecutar el script SampleMQM.
   - En Windows, el mandato para ejecutar el script SampleMQM es el siguiente:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\SampleMQM.bat
     ```
   - En AIX y Linux, el mandato para ejecutar el script SampleMQM es el siguiente:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

   donde *VÍA_INSTALACIÓN_MQ* es la ubicación en la que IBM WebSphere MQ está instalado.

   Se crea un gestor de colas denominado MQXR_SAMPLE_QM.
2. Escriba el mandato siguiente para ejecutar la primera parte del script MQTTV3Sample;
   - En Windows, en una línea de mandatos, escriba el mandato siguiente;

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -a subscribe
     ```
   - En AIX y Linux, en una ventana de shell, escriba el mandato siguiente:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscribe
     ```
3. Escriba el mandato siguiente para ejecutar la segunda parte del script MQTTV3Sample;
   - En Windows, en otra línea de mandatos, escriba el mandato siguiente:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -m "Hello from an MQTT v3
     application"
     ```
   - En AIX y Linux, en otra ventana de shell, escriba el mandato siguiente:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -m "Hello from an MQTT v3 application"
     ```

4. Para eliminar el gestor de colas que ha creado el script SampleMQM, puede ejecutar el script CleanupMQM utilizando el mandato;
   - En Windows, escriba el mandato siguiente;

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat
     ```

   - En AIX y Linux en otra ventana de shell, escriba el mandato siguiente;

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh
     ```

Resultados

El mensaje `Hello from an MQTT v3 application` , que ha escrito en la segunda ventana, será publicado por dicha aplicación y recibido por la aplicación en la primera ventana. La aplicación en la primera ventana la mostrará en la pantalla.

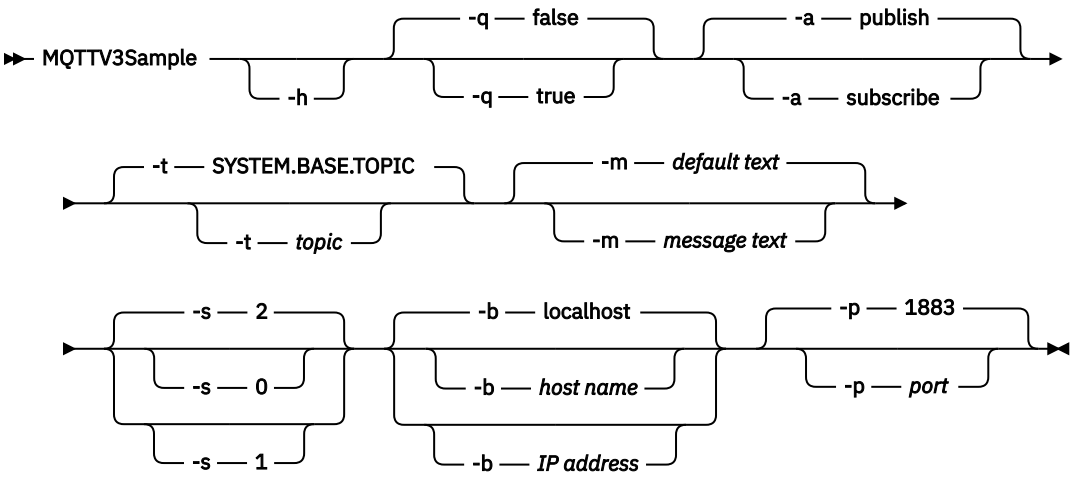
Programa MQTTV3Sample

Información de consulta sobre sintaxis y parámetros de ejemplo para el programa MQTTV3Sample .

Finalidad

El programa MQTTV3Sample se puede utilizar para publicar un mensaje y suscribirse a un tema.

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 texto de mensaje en lugar de enviar el texto de publicación predeterminado, "Hola desde una aplicación MQTT v3 ".

-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.

Ejecutar el programa MQTTV3Sample

Para suscribirse a un tema en Windows, utilice el mandato:

```
runMQTTV3Sample -a subscribe
```

Para publicar un mensaje en Windows, utilice el mandato:

```
runMQTTV3Sample
```

Para obtener más información sobre la ejecución de los scripts de muestra proporcionados, consulte "IBM WebSphere MQ Telemetry programas de ejemplo" en la página 482.

Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java

Los pasos para crear una aplicación cliente MQTT se describen en la guía de aprendizaje. Se explica cada línea de código. Al final de la tarea, habrá creado un publicador MQTT. Puede examinar las publicaciones utilizando WebSphere MQ Explorer.

Antes de empezar

Instale la característica WebSphere MQ Telemetry en un servidor que tenga instalado IBM WebSphere MQ Version 7.1 o posterior.

La aplicación cliente utiliza el paquete `com.ibm.mq.micro.client.mqttv3` en IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). El SDK forma parte de la instalación de IBM WebSphere MQ Telemetry . El cliente se conecta a la característica IBM WebSphere MQ Telemetry para intercambiar mensajes con IBM WebSphere MQ.

También debe instalar las actualizaciones de telemetría para que IBM WebSphere MQ Explorer Version 7.1 administre IBM WebSphere MQ Telemetry. Las actualizaciones forman parte de la instalación de IBM WebSphere MQ Telemetry .

Un cliente MQTT, que se ejecuta en Java SE, requiere la versión 6.0 de Java SE o posterior. IBM Java SE v6.0 forma parte de la instalación de IBM WebSphere MQ Version 7.1 . Se encuentra en *WebSphere MQ installation directory*`\java\jre`

Acerca de esta tarea

El ejemplo es una aplicación de publicación, `PubSync`. `PubSync` publica `Hello World` en el tema `MQTT Examples`y espera la confirmación de que la publicación se ha entregado al gestor de colas.

Al configurar una suscripción duradera en `MQTT Examples` , puede comprobar que la aplicación funciona.

El procedimiento utiliza Eclipse para desarrollar, compilar y ejecutar el cliente. Puede descargar Eclipse desde el sitio web del proyecto Eclipse en www.eclipse.org.

Para crear la aplicación, puede crear los archivos Java y compilarlos y ejecutarlos utilizando la línea de mandatos.

En un directorio nuevo, cree la vía de acceso del directorio `.\com\ibm\mq\id`. Cree dos archivos Java, `Example.java` y `PubSync.java`. Copie el código de "Código de ejemplo" en la página 489 en los archivos Java.

Compile el código Java utilizando el mandato,

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubSync.java com.ibm.mq.id.Example.java
```

Ejecute PubSync utilizando el mandato,

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubSync
```

Procedimiento

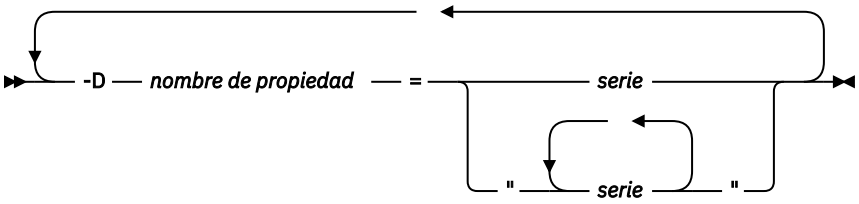
1. Cree un proyecto Java en Eclipse.
   a) **Archivo > Nuevo > Proyecto Java** y escriba un nombre de proyecto. Pulse **Siguiente**.

      Compruebe que el JRE está en la versión correcta o posterior. Java SE debe estar en 6.0 o posterior.
   b) En la página Valores de Java, pulse **Bibliotecas > Añadir archivos JAR externos ...**
   c) Vaya al directorio en el que ha instalado la carpeta WebSphere MQ Telemetry SDK. Localice la carpeta SDK`\clients\java` y seleccione todos los archivos `.jar` > **> Abrir > Finalizar**.
2. Instale el Javadoc del cliente MQTT.

   Con el Javadoc del cliente MQTT instalado, el editor Java proporciona ayuda con las clases v3 de MQTT.
   a) En el proyecto Java, abra **Explorador de paquetes > Bibliotecas referenciadas**. Pulse con el botón derecho `com.ibm.micro.client.mqttv3.jar` **> Propiedades**.
   b) En el navegador Propiedades pulse **Ubicación de Javadoc**.
   c) En la página Ubicación de Javadoc, pulse **URL de Javadoc > Examinar ...** y busque la carpeta *WMQ Installation directory*`\mqxr\SDK\clients\java\doc\javadoc` **> OK**.
   d) Pulse **Validar ... > Aceptar**

      Se le pedirá que abra un navegador para ver la documentación.
3. Cree la clase, PubSync, utilizando el asistente de clase Java.
   a) Pulse con el botón derecho del ratón en el proyecto Java que ha creado > **Nuevo** > **Clase**.
   b) Escriba el nombre del paquete, `com.ibm.mq.id`
   c) Escriba el nombre de clase, PubSync
   d) Compruebe el recuadro de apéndice de método, **public static void main (String [] args)**
4. Cree un archivo, `Example.java` en el paquete `com.ibm.mq.id`. Copie el código en Figura 89 en la página 491 en el archivo.

   Todos los parámetros utilizados en los ejemplos se establecen como propiedades. Puede alterar temporalmente los valores cambiando los valores predeterminados en `Example.java`, o proporcionando las propiedades como opciones en la línea de mandatos Java utilizando el parámetro `-D` :



El identificador de cliente utilizado en este ejemplo, y los ejemplos de “Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java” en la página 491 , es un nombre de usuario con el sufijo de una serie aleatoria.

5. Siga los pasos para crear el código o copie el código de Figura 88 en la página 490.

   Los pasos siguientes explican el código en `Pubsync.java`.

6. Cree un bloque `try-catch` .

   ```
   try { ...
    } catch (Exception e) {
     e.printStackTrace();
   }
   ```

   El cliente MQTT genera `MqttException`, `MqttPersistenceException` o `MqttSecurityException`. `MqttPersistenceException` y `MqttSecurityException` son subclases de `MqttException`.

   Utilice el método `MqttException.getReasonCode` para averiguar el motivo de la excepción. Si se genera un `MqttPersistenceException` o `MqttSecurityException` , utilice el método `getCause` para devolver la excepción throwable subyacente.

7. Cree una nueva instancia de `MqttClient` .

   ```
   MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
   ```

   Proporcione al cliente una dirección de servidor, que se utilizará posteriormente para conectarse a WebSphere MQ. Establezca el identificador de cliente para denominar el cliente.

   - Opcionalmente, puede proporcionar una implementación de la interfaz `MqttClientPersistence` para sustituir la implementación predeterminada. La implementación predeterminada de `MqttPersistence` almacena QoS 1 y 2 mensajes en espera de entrega como archivos; consulte “Persistencia de mensajes en clientes MQTT” en la página 545.
   - El puerto TCP/IP de IBM WebSphere MQ predeterminado para MQTT es 1883. Para SSL, es 8883. En el ejemplo, la dirección predeterminada se establece en `tcp://localhost:1883`.
   - Normalmente, es importante poder identificar un cliente físico específico utilizando el identificador de cliente. El identificador de cliente debe ser exclusivo entre todos los clientes que se conectan a un servidor; consulte “Identificador de cliente” en la página 541. El uso del mismo identificador de cliente que una instancia anterior indica que la instancia actual es una instancia del mismo cliente. Si duplica un identificador de cliente en dos clientes en ejecución, se genera una excepción en ambos clientes y un cliente termina.
   - La longitud del identificador de cliente está limitada a 23 bytes. Se genera una excepción, si se supera la longitud. El identificador de cliente sólo debe contener caracteres permitidos en un nombre de gestor de colas; por ejemplo, sin guiones ni espacios.
   - Hasta que llame al método `MqttClient.connect` , no tendrá lugar ningún proceso de mensajes.

   Utilice el objeto de cliente para publicar y suscribir temas y recuperar información sobre publicaciones que todavía no se han entregado.

8. Cree un tema en el que publicar.

   ```
   MqttTopic topic = client.getTopic(Example.topicString);
   ```

   Una serie de tema está limitada a 64 K bytes, lo que supera la longitud máxima de una serie de tema IBM WebSphere MQ . De lo contrario, una serie de tema sigue las mismas reglas que las series de tema de WebSphere MQ ; consulte Series de tema. El ejemplo establece la serie de tema `MQTT Examples`.

9. Cree un mensaje de publicación.

   ```
   MqttMessage message = new MqttMessage(Example.publication.getBytes());
   ```

La serie `"Hello World"` se convierte en una matriz de bytes y se utiliza para crear un `MqttMessage`.

- Una carga útil de mensaje MQTT es siempre una matriz de bytes. El método `getBytes` convierte un objeto de serie a UTF-8. `MqttMessage` tiene un método práctico `toString` para devolver la carga útil de mensaje como una serie. Equivale a `new string(message.getPayload)`
- Se envía un mensaje de publicación al gestor de colas con una cabecera RFH2 y los datos del mensaje se envían como un mensaje `jms-bytes` .
- El objeto de mensaje tiene calidad de servicio y atributos retenidos. La calidad de servicio (QoS) determina la fiabilidad con la que se transfiere el mensaje entre el cliente MQTT y el gestor de colas; consulte “Calidades de servicio proporcionadas por un cliente MQTT” en la página 549. El atributo retenido controla si el gestor de colas almacena una publicación para futuros suscriptores. Si una publicación no se retiene, sólo se envía a los suscriptores actuales; consulte “Publicaciones retenidas y clientes MQTT” en la página 551. Los valores predeterminados de `MqttMessage` son: "Los mensajes se entregan al menos una vez y no se conservan."

10. Conéctese al servidor.

```
client.connect();
```

El ejemplo se conecta al servidor utilizando las opciones de conexión predeterminadas. Una vez que se conecte, puede empezar a publicar. Las opciones de conexión predeterminadas son:

- Se envía un pequeño mensaje "keep-alive" cada 15 segundos para evitar que se cierre la conexión TCP/IP.
- La sesión se inicia sin comprobar la finalización de publicaciones anteriores.
- El intervalo entre intentar volver a enviar un mensaje es de 15 segundos.
- No se crea ningún mensaje de última voluntad y testamento para la conexión.
- La SocketFactory estándar se utiliza para crear la conexión.

Cambie las opciones de conexión creando un objeto `ConnectionOptions` y pasándolo como parámetro adicional a `client.connect`.

11. Publicar.

```
MqttDeliveryToken token = topic.publish(message);
```

El ejemplo envía la publicación "Hello World" sobre el tema "Ejemplos de MQTT" al gestor de colas.

- Cuando se devuelve el método `publish` , el mensaje se transfiere de forma segura al cliente MQTT, pero todavía no se transfiere al servidor. Si el mensaje tiene QoS 1 o 2, el mensaje se almacena localmente, en caso de que el cliente falle antes de que se complete la entrega.
- `publish` devuelve una señal de entrega, que se utiliza para comprobar si se ha recibido todavía un acuse de recibo del servidor.

12. Espere el acuse de recibo del servidor.

```
token.waitForCompletion(Example.timeout);
```

El ejemplo PubSync espera un acuse de recibo del servidor, lo que confirma que el mensaje se ha entregado.

- Sin el tiempo de espera, el cliente esperaría indefinidamente. La tarea “Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java” en la página 491 muestra cómo recibir confirmaciones sin esperar utilizando un objeto de devolución de llamada.

13. Desconecte el cliente del servidor.

```
client.disconnect();
```

El cliente se desconecta del servidor y espera a que finalicen los métodos `MqttCallback` que se están ejecutando. A continuación, espera hasta 30 segundos para finalizar cualquier trabajo restante. Puede especificar un tiempo de espera de inmovilización como parámetro adicional.

14. Guardar cambios en `PubSync.java` y `Example.Java`

    Eclipse compila automáticamente Java. Ahora está listo para ver los resultados ejecutando el programa.

Resultados

Para ver las publicaciones que utilizan WebSphere MQ, cree un tema, una cola y una suscripción duradera, todos ellos denominados `"MQTTExampleTopic"` utilizando el script en Figura 87 en la página 489. Ejecute el cliente para publicar en el tema `MQTT Examples` y, a continuación, ejecute el programa de ejemplo **amqsbcg** para examinar las publicaciones en la cola `MQTTExamples` .

1. Inicie un gestor de colas e inicie su servicio de telemetría (MQXR) en ejecución. Asegúrese de que la dirección TCP/IP y el puerto configurados para el canal de telemetría coinciden con los valores que utiliza en la aplicación MQTT.
2. Configure una suscripción duradera creando el script de mandatos `mqttexamples.txt` y ejecutarlo utilizando **runmqsc:**

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Figura 87. `mqttExampleTopic.txt`

Para ejecutar el script en Windows, escriba el mandato:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Ejecute el cliente como una aplicación Java desde Eclipse, o ejecutando Java en una ventana de mandatos:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Nota: La ventana de mandatos debe estar abierta en el directorio que contiene la vía de acceso, `com\ibm\mq\id`.

4. Examine los resultados utilizando WebSphere MQ Explorer o ejecute el mandato:

```
amqsbcg MQTTExampleQueue queue manager name
```

Código de ejemplo

PubSync.java es un listado completo del código descrito en Procedimiento. Modifique la clase `Example` en Figura 89 en la página 491 para alterar temporalmente los parámetros predeterminados utilizados en PubSync.java.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      client.connect();
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
          + " seconds for publication of \"" + message.toString()
          + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName()
          + "\" for client instance: \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 88. PubSync.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figura 89. Example.java

Conceptos relacionados

Aplicaciones de publicación/suscripción de MQTT

Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java

En esta tarea, siga una guía de aprendizaje para modificar la primera aplicación de publicación. Las modificaciones permiten que la aplicación envíe publicaciones sin esperar confirmaciones de entrega. Los acuses de recibo de entrega los recibe una clase de devolución de llamada que crea.

Antes de empezar

Instale la característica WebSphere MQ Telemetry en un servidor que tenga instalado IBM WebSphere MQ Version 7.1 o posterior.

La aplicación cliente utiliza el paquete `com.ibm.mq.micro.client.mqttv3` en IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). El SDK forma parte de la instalación de IBM WebSphere MQ Telemetry . El cliente se conecta a la característica IBM WebSphere MQ Telemetry para intercambiar mensajes con IBM WebSphere MQ.

También debe instalar las actualizaciones de telemetría para que IBM WebSphere MQ Explorer Version 7.1 administre IBM WebSphere MQ Telemetry. Las actualizaciones forman parte de la instalación de IBM WebSphere MQ Telemetry .

Un cliente MQTT, que se ejecuta en Java SE, requiere la versión 6.0 de Java SE o posterior. IBM Java SE v6.0 forma parte de la instalación de IBM WebSphere MQ Version 7.1 . Se encuentra en *WebSphere MQ installation directory*\java\jre

Acerca de esta tarea

El ejemplo es una aplicación de publicación, PubAsync. PubAsync publica `Hello World` sobre el tema `MQTT Examples`, sin esperar a que se confirme que la publicación se ha entregado al gestor de colas. Los acuses de recibo de entrega se reciben en una clase de devolución de llamada, `CallBack`.

Al configurar una suscripción duradera en `MQTT Examples` , puede comprobar que la aplicación funciona.

El procedimiento utiliza Eclipse para desarrollar, compilar y ejecutar el cliente. Puede descargar Eclipse desde el sitio web del proyecto Eclipse en www.eclipse.org.

Los pasos de Procedimiento modifican la aplicación PubSync`.java` en "Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java" en la página 485.

De forma alternativa, puede copiar el código, "Código de ejemplo" en la página 494, en un directorio nuevo `.\com\ibm\mq\id`. Cree tres archivos Java, `Example.java`, `CallBack.javay` PubAsync`.java`. Compile los ejemplos utilizando el mandato,

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsync.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Example.java
```

Ejecute PubAsync utilizando el mandato,

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsync.class
```

Procedimiento

1. En el paquete `com.ibm.mq.id` , cree un archivo, `CallBack.java`. Copie el código en Figura 92 en la página 495 en el archivo.

   `Callback.java` implementa la interfaz `MqttCallBack` . En el ejemplo, un constructor adicional inicializa la devolución de llamada con algunos datos de instancia.
2. En el paquete `com.ibm.mq.id` , pulse con el botón derecho del ratón en PubSync`.java` y cópielo. Péguelo en el mismo paquete, renombrándolo como PubAsync.
3. Justo antes de la línea de código `client.connect();` , cree una instancia de la clase `CallBack` , pasando el identificador de cliente.

   ```
   CallBack callback = new CallBack(Example.clientId);
   client.setCallback(callback);
   ```

   - La clase `CallBack` implementa `MqttCallBack`. Se necesita una instancia de devolución de llamada, por identificador de cliente. En este ejemplo, el constructor pasa el identificador de cliente para guardar como datos de instancia. Se utiliza en la devolución de llamada para identificar qué instancia de la devolución de llamada se ha iniciado.
   - Debe implementar tres métodos en la clase de devolución de llamada:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Recibe una publicación a la que se ha suscrito.

     `public void connectionLost(Throwable cause)`
     Se llama cuando se pierde la conexión.

public void deliveryComplete(MqttDeliveryToken token))
Se llama cuando se recibe una señal de entrega para un mensaje QoS 1 o 2 que se ha publicado.

- `MqttClient.connect`activa la devolución de llamada.

4. Desconectar el cliente
   a) Elimine la sentencia que contiene la expresión `token.waitForCompletion` .
   El hilo principal continúa sin esperar a que se entregue la publicación.
   b) Pruebe si el cliente ya está desconectado.
   El cliente MQTT se desconecta después de un error devuelto al método `lostConnection` en `MqttCallback`, o la aplicación cliente podría desconectarse. Pruebe si hay una conexión abierta.
   c) Utilice la constante, `Example.quiesceTimeout`, para establecer el tiempo máximo para desactivar temporalmente el cliente.

```
if (client.isConnected())
    client.disconnect(Example.quiesceTimeout);
```

El cliente finaliza cuando se cumple una combinación de las tres condiciones siguientes:

a. Se ha llamado a la devolución de llamada para todos los mensajes que se han publicado en esta sesión, o si la sesión se ha reiniciado, en sesiones anteriores.
b. Los mensajes están en curso y el intervalo de inmovilización ha caducado. De forma predeterminada, el intervalo de desactivación temporal es de 30 segundos. Puede cambiar el tiempo de espera de desactivación temporal pasando el número de milisegundos a esperar como parámetro de `client.disconnect`.
c. Se ha llamado a `client.disconnect` después de que el cliente publicara algunos mensajes y los hubiera puesto en cola, pero antes de que se enviaran los mensajes. Los mensajes en cola todavía no están en curso. Si la sesión es reiniciable, los mensajes se reenvían cuando se reinicia la sesión.

Resultados

Para ver las publicaciones que utilizan WebSphere MQ, cree un tema, una cola y una suscripción duradera, todos ellos denominados `"MQTTExampleTopic"` utilizando el script en Figura 90 en la página 493. Ejecute el cliente para publicar en el tema `MQTT Examples` y, a continuación, ejecute el programa de ejemplo **amqsbcg** para examinar las publicaciones en la cola `MQTTExamples` .

1. Inicie un gestor de colas e inicie su servicio de telemetría (MQXR) en ejecución. Asegúrese de que la dirección TCP/IP y el puerto configurados para el canal de telemetría coinciden con los valores que utiliza en la aplicación MQTT.
2. Configure una suscripción duradera creando el script de mandatos `mqttexamples.txt` y ejecutarlo utilizando **runmqsc:**

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Figura 90. mqttExampleTopic.txt

Para ejecutar el script en Windows, escriba el mandato:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Ejecute el cliente como una aplicación Java desde Eclipse, o ejecutando Java en una ventana de mandatos:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Nota: La ventana de mandatos debe estar abierta en el directorio que contiene la vía de acceso, `com\ibm\mq\id`.

4. Examine los resultados utilizando WebSphere MQ Explorer o ejecute el mandato:

```
amqsbcg MQTTExampleQueue queue manager name
```

Código de ejemplo

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubAsync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      client.connect();
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 91. PubAsync.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 92. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figura 93. Example.java

Creación de un publicador asíncrono recuperable para MQ Telemetry Transport utilizando Java

En esta tarea, siga una guía de aprendizaje para modificar la aplicación de publicador asíncrono. Las modificaciones permiten a la aplicación completar la entrega de publicaciones que no se reconocieron la última vez que se ejecutó el cliente.

Antes de empezar

Instale la característica WebSphere MQ Telemetry en un servidor que tenga instalado IBM WebSphere MQ Version 7.1 o posterior.

La aplicación cliente utiliza el paquete `com.ibm.mq.micro.client.mqttv3` en IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). El SDK forma parte de la instalación de IBM WebSphere MQ Telemetry . El cliente se conecta a la característica IBM WebSphere MQ Telemetry para intercambiar mensajes con IBM WebSphere MQ.

También debe instalar las actualizaciones de telemetría para que IBM WebSphere MQ Explorer Version 7.1 administre IBM WebSphere MQ Telemetry. Las actualizaciones forman parte de la instalación de IBM WebSphere MQ Telemetry .

Un cliente MQTT, que se ejecuta en Java SE, requiere la versión 6.0 de Java SE o posterior. IBM Java SE v6.0 forma parte de la instalación de IBM WebSphere MQ Version 7.1 . Se encuentra en *WebSphere MQ installation directory*\java\jre

Acerca de esta tarea

El ejemplo es una aplicación de publicación, PubAsyncRestartable. PubAsyncRestartable publica Hello World sobre el tema MQTT Examples, sin esperar a la confirmación de que la publicación se ha entregado al gestor de colas. Los acuses de recibo de entrega se reciben en una clase de devolución de llamada, CallBack. Se pueden examinar las señales de entrega para las publicaciones que no se han completado en una instancia anterior. También los procesa la clase de devolución de llamada.

Al configurar una suscripción duradera en MQTT Examples , puede comprobar que la aplicación funciona.

El procedimiento utiliza Eclipse para desarrollar, compilar y ejecutar el cliente. Puede descargar Eclipse desde el sitio web del proyecto Eclipse en www.eclipse.org.

Los pasos de Procedimiento modifican la aplicación PubAsync.java en "Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java" en la página 491.

De forma alternativa, puede copiar el código, "Código de ejemplo" en la página 500, en un directorio nuevo .\com\ibm\mq\id. Cree tres archivos Java, Example.java, CallBack.javay PubAsyncRestartable.java. Compile los ejemplos utilizando el mandato,

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsyncRestartable.java com.ibm.mq.id.CallBack.java
com.ibm.mq.id.Example.java
```

Ejecute PubAsyncRestartable utilizando el mandato,

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsyncRestartable.class
```

Procedimiento

1. En el paquete com.ibm.mq.id , pulse con el botón derecho del ratón en PubAsync.java y cópielo. Péguelo en el mismo paquete, renombrándolo como PubAsyncRestartable.
2. Cree un identificador de cliente reutilizable.

   ```
   Example.clientId = String.format(
       "%-23.23s",
       (System.getProperty("user.name") + "_" + (System.getProperty(
           "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
   ```

   Figura 94. Identificador de cliente reutilizable

   Las aplicaciones de "Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java" en la página 485 y "Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java" en la página 491 han utilizado un nuevo identificador de cliente para cada conexión de cliente. Para un publicador o suscriptor reiniciable, debe utilizar el mismo identificador de cliente cada vez que se conecte el cliente, pero distintos clientes deben utilizar identificadores diferentes; consulte "Identificador de cliente" en la página 541. El identificador de cliente reutilizable se crea a partir del nombre de usuario y el nombre de la clase. Está limitado a 23 bytes de longitud. Sólo debe tener caracteres que sean válidos en los nombres de objeto del gestor de colas. El código elimina los guiones que puedan haberse insertado.
3. La QoS del mensaje se establece en 2 en lugar del valor predeterminado, 1, para evitar mensajes duplicados.

```
message.setQos(Example.QoS);
```

Debe cambiar el valor de `Example.QoS` a 2o pasar la propiedad `QoS` como argumento utilizando la opción `-DQoS=2` en la línea de mandatos de Java.

4. Cree un objeto `MqttConnectOptions` y establezca su atributo `cleanSession` en `false`.

   a) Cree un objeto `MqttConnectOptions` .

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   ```

   `conOptions` es un parámetro de opción en el constructor `MqttClient` .

   b) Establezca el atributo clearSession .

   ```
   conOptions.setCleanSession(Example.cleanSession);
   ```

   De forma predeterminada, el parámetro `Example.cleanSession` se establece en `true`, coincidiendo con el valor predeterminado de `MqttConnectionOptions.cleanSession`.

   Cuando se reinicia `PubAsyncRestartable` , puede empezar con una "sesión limpia" y borrar las señales de entrega pendientes para los mensajes de QoS 1 o 2.

   Establezca `Example.cleanSession` en `false` para mantener todas las señales de entrega pendientes. La clase `MqttCallBack` procesa las señales cuando se vuelve a conectar el cliente.

5. Si la sesión se está reiniciando, recupere las señales de entrega pendientes e imprima su contenido.

```
if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance "
            + client.getClientId());
```

6. Pase el parámetro `conOptions` al constructor `MqttClient` .

```
client.connect(conOptions);
```

7. Al desconectar, establezca un intervalo máximo de desconexión.

```
client.disconnect(Example.timeout);
```

Para poder mostrar las señales de entrega pendientes que se están procesando, una instancia anterior debe finalizar sin completar la entrega. Para ejecutar el ejemplo con la posibilidad de no reconocer publicaciones antes de que finalice `PubAsyncRestartable` , establezca `Example.timeout` en 0.

Resultados

Para ver las publicaciones que utilizan WebSphere MQ, cree un tema, una cola y una suscripción duradera, todos ellos denominados `"MQTTExampleTopic"` utilizando el script en Figura 95 en la página 499. Ejecute el cliente para publicar en el tema `MQTT Examples` y, a continuación, ejecute el programa de ejemplo **amqsbcg** para examinar las publicaciones en la cola `MQTTExamples` .

1. Inicie un gestor de colas e inicie su servicio de telemetría (MQXR) en ejecución. Asegúrese de que la dirección TCP/IP y el puerto configurados para el canal de telemetría coinciden con los valores que utiliza en la aplicación MQTT.
2. Configure una suscripción duradera creando el script de mandatos `mqttexamples.txt` y ejecutarlo utilizando **runmqsc:**

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Figura 95. mqttExampleTopic.txt

Para ejecutar el script en Windows, escriba el mandato:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Ejecute el cliente como una aplicación Java desde Eclipse, o ejecutando Java en una ventana de mandatos:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Nota: La ventana de mandatos debe estar abierta en el directorio que contiene la vía de acceso, `com\ibm\mq\id`.

4. Examine los resultados utilizando WebSphere MQ Explorer o ejecute el mandato:

```
amqsbcg MQTTExampleQueue queue manager name
```

Código de ejemplo

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
import com.ibm.micro.client.mqttv3.MqttDeliveryToken;
import com.ibm.micro.client.mqttv3.MqttMessage;
import com.ibm.micro.client.mqttv3.MqttTopic;
public class PubAsyncRestartable {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + (System.getProperty(
            "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance \""
            + client.getClientId() + "\"");
      client.connect(conOptions);
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 96. PubAsyncRestartable.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 97. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figura 98. Example.java

Creación de un suscriptor para MQ Telemetry Transport utilizando Java

En esta tarea, siga una guía de aprendizaje para crear una aplicación de suscriptor. El suscriptor crea una suscripción a un tema y recibe publicaciones para la suscripción.

Antes de empezar

Instale la característica WebSphere MQ Telemetry en un servidor que tenga instalado IBM WebSphere MQ Version 7.1 o posterior.

La aplicación cliente utiliza el paquete `com.ibm.mq.micro.client.mqttv3` en IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). El SDK forma parte de la instalación de IBM WebSphere MQ Telemetry . El cliente se conecta a la característica IBM WebSphere MQ Telemetry para intercambiar mensajes con IBM WebSphere MQ.

También debe instalar las actualizaciones de telemetría para que IBM WebSphere MQ Explorer Version 7.1 administre IBM WebSphere MQ Telemetry. Las actualizaciones forman parte de la instalación de IBM WebSphere MQ Telemetry .

Un cliente MQTT, que se ejecuta en Java SE, requiere la versión 6.0 de Java SE o posterior. IBM Java SE v6.0 forma parte de la instalación de IBM WebSphere MQ Version 7.1 . Se encuentra en *WebSphere MQ installation directory*\java\jre

Acerca de esta tarea

El ejemplo es una aplicación de suscriptor, Subscribe. Subscribe crea un tema de suscripción, MQTT Examples, y espera las publicaciones en la suscripción durante 30 segundos.

Un suscriptor puede crear una suscripción y esperar publicaciones. También puede recibir publicaciones enviadas a una suscripción creada anteriormente, para el mismo identificador de cliente. El atributo booleano MqttConnectionOptions.cleanSession controla si las publicaciones enviadas anteriormente se reciben o no; consulte "Suscripciones" en la página 552.

Puede utilizar los programas de ejemplo de publicación para crear publicaciones o utilizar el explorador de WebSphere MQ para crear una publicación de prueba en el tema MQTT Examples .

El procedimiento utiliza Eclipse para desarrollar, compilar y ejecutar el cliente. Puede descargar Eclipse desde el sitio web del proyecto Eclipse en www.eclipse.org.

Las instrucciones de Procedimiento presuponen que ya ha creado el paquete com.ibm.mq.id en una de las tareas anteriores y que se ha copiado en las clases Example.java y Callback.java .

Procedimiento

1. Cree la clase, Subscribe en el paquete com.ibm.mq.id.
2. Cree un identificador de cliente reutilizable.

```
Example.clientId = String.format(
    "%-23.23s",
    (System.getProperty("user.name") + "_" + (System.getProperty(
        "clientId", "Subscribe."))).trim()).replace('-', '_');
```

Figura 99. Identificador de cliente reutilizable

Las aplicaciones de "Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java" en la página 485 y "Creación de un publicador asíncrono para MQ Telemetry Transport utilizando Java" en la página 491 han utilizado un nuevo identificador de cliente para cada conexión de cliente. Para un publicador o suscriptor reiniciable, debe utilizar el mismo identificador de cliente cada vez que se conecte el cliente, pero distintos clientes deben utilizar identificadores diferentes; consulte "Identificador de cliente" en la página 541. El identificador de cliente reutilizable se crea a partir del nombre de usuario y el nombre de la clase. Está limitado a 23 bytes de longitud. Sólo debe tener caracteres que sean válidos en los nombres de objeto del gestor de colas. El código elimina los guiones que puedan haberse insertado.

3. Cree un bloque try-catch .

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

El cliente MQTT genera MqttException, MqttPersistenceException o MqttSecurityException. MqttPersistenceException y MqttSecurityException son subclases de MqttException.

Utilice el método MqttException.getReasonCode para averiguar el motivo de la excepción. Si se genera un MqttPersistenceException o MqttSecurityException , utilice el método getCause para devolver la excepción throwable subyacente.

4. Cree una nueva instancia de MqttClient .

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

Proporcione al cliente una dirección de servidor, que se utilizará posteriormente para conectarse a WebSphere MQ. Establezca el identificador de cliente para denominar el cliente.

- Opcionalmente, puede proporcionar una implementación de la interfaz `MqttClientPersistence` para sustituir la implementación predeterminada. La implementación predeterminada de `MqttPersistence` almacena QoS 1 y 2 mensajes en espera de entrega como archivos; consulte "Persistencia de mensajes en clientes MQTT" en la página 545.
- El puerto TCP/IP de IBM WebSphere MQ predeterminado para MQTT es 1883. Para SSL, es 8883. En el ejemplo, la dirección predeterminada se establece en `tcp://localhost:1883`.
- Normalmente, es importante poder identificar un cliente físico específico utilizando el identificador de cliente. El identificador de cliente debe ser exclusivo entre todos los clientes que se conectan a un servidor; consulte "Identificador de cliente" en la página 541. El uso del mismo identificador de cliente que una instancia anterior indica que la instancia actual es una instancia del mismo cliente. Si duplica un identificador de cliente en dos clientes en ejecución, se genera una excepción en ambos clientes y un cliente termina.
- La longitud del identificador de cliente está limitada a 23 bytes. Se genera una excepción, si se supera la longitud. El identificador de cliente sólo debe contener caracteres permitidos en un nombre de gestor de colas; por ejemplo, sin guiones ni espacios.
- Hasta que llame al método `MqttClient.connect` , no tendrá lugar ningún proceso de mensajes.

Utilice el objeto de cliente para publicar y suscribir temas y recuperar información sobre publicaciones que todavía no se han entregado.

5. Justo antes de la línea de código `client.connect();` , cree una instancia de la clase `CallBack` , pasando el identificador de cliente.

```
CallBack callback = new CallBack(Example.clientId);
client.setCallback(callback);
```

   - La clase `CallBack` implementa `MqttCallBack`. Se necesita una instancia de devolución de llamada, por identificador de cliente. En este ejemplo, el constructor pasa el identificador de cliente para guardar como datos de instancia. Se utiliza en la devolución de llamada para identificar qué instancia de la devolución de llamada se ha iniciado.
   - Debe implementar tres métodos en la clase de devolución de llamada:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Recibe una publicación a la que se ha suscrito.

     `public void connectionLost(Throwable cause)`
     Se llama cuando se pierde la conexión.

     `public void deliveryComplete(MqttDeliveryToken token))`
     Se llama cuando se recibe una señal de entrega para un mensaje QoS 1 o 2 que se ha publicado.
   - `MqttClient.connect`activa la devolución de llamada.
6. Cree un objeto `MqttConnectOptions` y establezca su atributo `cleanSession` .
   a) Cree un objeto `MqttConnectOptions` .

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

      `conOptions` es un parámetro de opción en el constructor `MqttClient` .
   b) Establezca el atributo clearSession .

```
conOptions.setCleanSession(Example.cleanSession);
```

      De forma predeterminada, el parámetro `Example.cleanSession` se establece en `true`, coincidiendo con el valor predeterminado de `MqttConnectionOptions.cleanSession`.

Si utiliza el `MqttConnectOptions` predeterminado, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar el cliente, se eliminan las

suscripciones antiguas del clientes 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 cree el cliente se añaden a todas las suscripciones que ya existían del mismo antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otra forma de entender la forma en que el atributo `cleanSession` afecta a las suscripciones es considerarlo 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.

La modalidad `cleanSession` debe definirse antes de la conexión y dura la sesión completa. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de `cleanSession=false` a `cleanSession=true`, se descartan todas las suscripciones previas del cliente y cualquier publicación que no se hayan recibido.

7. Pase el parámetro `conOptions` al constructor `MqttClient` .

```
client.connect(conOptions);
```

8. Cree una suscripción.

```
      client.subscribe(Example.topicString, Example.QoS);
```

El ejemplo utiliza un método `MqttClient.subscribe` que pasa un filtro de tema con una opción `QoS` . El método `MqttClient.subscribe` tiene cuatro firmas y puede pasar matrices de filtros de suscripción, así como un único filtro.

El ejemplo utiliza la serie de tema utilizada por los ejemplos de publicación como filtro de tema, por lo que recibe las publicaciones que crean.

Cada vez que ejecuta el ejemplo, `subscribe.java`, crea una suscripción. Si no cambia Example.topicString, vuelve a crear la misma suscripción. Si se vuelve a crear una suscripción, no da como resultado dos suscripciones idénticas. Un cliente no recibe copias duplicadas de publicaciones que coincidan con una suscripción idéntica.

Las suscripciones se describen en “Suscripciones” en la página 552y los filtros en “Series de tema y filtros de tema en clientes MQTT” en la página 554.

9. Espere a que lleguen algunas publicaciones y, a continuación, desconecte el cliente.

```
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
```

Las publicaciones se reciben mediante la implementación del método `MqttCallback.messageArrived` .

La aplicación de suscripción no ha publicado ningún mensaje y, por lo tanto, no espera ninguna señal de entrega. `client.disconnect` tiene lugar sin ningún retardo.

Código de ejemplo

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
public class Subscribe {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + System.getProperty("clientId",
            "Subscribe."))).trim();
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      client.connect(conOptions);
      System.out.println("Subscribing to topic \"" + Example.topicString
          + "\" for client instance \"" + client.getClientId()
          + "\" using QoS " + Example.QoS + ". Clean session is "
          + Example.cleanSession);
      client.subscribe(Example.topicString, Example.QoS);
      System.out.println("Going to sleep for " + Example.sleepTimeout / 1000
          + " seconds");
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
      System.out.println("Finished");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 100. Subscribe.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figura 101. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figura 102. Example.java

Conceptos relacionados

Aplicaciones de publicación/suscripción de MQTT

Autenticación de un cliente Java MQTT utilizando JAAS

Aprenda a autenticar un cliente utilizando JAAS. Modifique el programa de ejemplo `JAASLoginModule.java` y el programa Java de ejemplo `PubSync.java`. Configure un canal de telemetría para requerir la autenticación JAAS y ejecute el publicador modificado, comprobando su `nombre de usuario` y `contraseña` utilizando JAAS.

Antes de empezar

Se supone que ha instalado los archivos jar del cliente MQTT v3 , Javadoc, Eclipse, ha configurado canales de telemetría y ha codificado y ejecutado PubSync.java antes de realizar esta tarea. Tiene un espacio de trabajo de Eclipse que incluye una versión en ejecución de PubSync.java.

La tarea se escribe para Windows. Cambie las vías de acceso de directorio para Linux.

Acerca de esta tarea

La tarea se basa en la modificación de la clase JAASLoginModule de ejemplo en *WMQ Installation directory*\mqxr\samples\JAASLoginModule.java para crear MyLogin.java. En la tarea también modifica el código de ejemplo, PubSync.java en "Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java" en la página 485 para establecer un username y password. Como prueba, MyLogin.java acepta o rechaza aleatoriamente nombre_usuario y contraseña.

Los pasos de la tarea se escriben como un ejercicio de programación. Debe adaptar el procedimiento para realizar la autenticación real en un entorno de producción.

En una explicación típica de cómo programar la autenticación JAAS , se presupone que el módulo de inicio de sesión está autenticando el contexto que ha cargado JAAS. Cuando el servicio de telemetría (MQXR) llama a JAAS, el contexto que ha cargado JAAS es el servicio de telemetría (MQXR). No hay ningún punto en la autenticación del contexto de servicio de telemetría (MQXR); siempre es mqm. En su lugar, el servicio de telemetría (MQXR) configura el nombre de usuario y la contraseña del cliente para que estén disponibles para la clase de módulo de inicio de sesión. El nombre de usuario y la contraseña se pasan al módulo de inicio de sesión utilizando dos devoluciones de llamada.

```
javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
callbacks[0] =
      new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] =
      new javax.security.auth.callback.PasswordCallback("PasswordCallback", false);
callbackHandler.handle(callbacks);
String username =
      ((javax.security.auth.callback.NameCallback) callbacks[0]).getName();
char[] password =
      ((javax.security.auth.callback.PasswordCallback) callbacks[1]).getPassword();
```

El nombre de usuario y la contraseña del cliente son la única información sobre el cliente que está disponible para el módulo de inicio de sesión.

Procedimiento

1. Cree dos paquetes, samples y security.jaas en el mismo proyecto Java que PubSync.java.

   El paquete samples sólo se utiliza como referencia. Realice los cambios de código en el paquete security.jaas .
2. Importe JAASLoginModule.java y JAASPrincipal.java en ambos paquetes.

   Si es necesario, refactorice las sentencias de paquete en el fuente Java para eliminar los errores de compilación.
3. Refactorizar el nombre de clase, JAASLoginModule, en el paquete security.jaas a MyLogin
4. En MyLogin.java, sustituya parte del código en el método login para mostrar el módulo funcionando.
   a) Sustituya el código:

   ```
   // Accept everything.
   if (true)
     loggedIn = true;
   else
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   ```

   b) Con el código:

   ```
   // login half the users randomly
   PrintWriter pw = new PrintWriter(new FileWriter(System.getProperty("user.dir")
           + "\\MyLogin.log", true));
   pw.println("Called JAASLogin.login at "
           + System.getProperty("publication", "Hello World "
           + String.format("%tc", System.currentTimeMillis())));
   if (Math.random() < 0.5)
     loggedIn = true;
   pw.println("Username: \"" + username + "\", Password: \""
       + String.valueOf(password) + "\" loggedIn: " + loggedIn);
   ```

```
pw.close();
if (!loggedIn)
  throw new javax.security.auth.login.FailedLoginException("Login failed");
principal= new JAASPrincipal(username);
```

El origen completo de `MyLogin.java` se encuentra en Figura 105 en la página 511. El origen de `JAASPrincipal.java`, con el nombre de paquete refactorizado en `security.jaas` está en Figura 106 en la página 512.

5. Establezca la vía de acceso de clases en `service.env` para que apunte al directorio que contiene la vía de acceso a `security/jaas/MyLogin.class` y `security/jaas/JAASPrincipal.class`.

   ```
   CLASSPATH=C:\WMQTelemetryApps\MQTTSecureExamples\bin
   ```

   Consulte Configuración de canal de telemetría JAAS para obtener información sobre cómo utilizar `service.env` para pasar una vía de acceso de clases a un servicio WebSphere MQ .

6. Añada una stanza de módulo de inicio de sesión a `jaas.config`.

   ```
   MyLoginExample {
     security.jaas.MyLogin required debug=true;
     };
   ```

   Consulte Configuración de canal de telemetría JAAS para obtener información sobre cómo utilizar `jaas.config` definir un módulo de inicio de sesión JAAS .

7. Añada un canal de telemetría utilizando el asistente **Nuevo canal de telemetría** en WebSphere MQ Explorer, configurando el canal para que requiera autenticación JAAS . Consulte la stanza `MyLoginExample` .

   Por ejemplo, adapte la información que escriba en el asistente desde esta stanza en el archivo `mqxr_win.properties` . Si está trabajando en Linux , el archivo se denomina `mqxr_unix.properties`. No edite el archivo de propiedades de telemetría directamente; utilice el asistente.

   ```
   com.ibm.mq.MQXR.channel/JAASMCAUser: \
   com.ibm.mq.MQXR.Port=1884;\
   com.ibm.mq.MQXR.JAASConfig=MyLoginExample;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

   Nota: Si modifica alguno de los parámetros de canal de telemetría, o modifica la clase `security.jaas.Mylogin` , debe detener y reiniciar el servicio de telemetría (MQXR). Sólo cuando reinicie el servicio, los cambios entrarán en vigor.

8. Haga una copia de `PubSync.java` en el paquete `com.ibm.mq.id` y denomine la copia `PubSyncJAAS.java`.

   Consulte “Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java” en la página 485 para ver los pasos para crear PubSync.java en el paquete `com.ibm.mq.id` .

9. Establezca `MqttConnectOptions.username` y `MqttConnectOptions.password` en el programa `PubSyncJAAS.java` y pase `MqttConnectOptions` como parámetro de `MqttClient.connect`.

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setUserName(Example.username);
   conOptions.setPassword(Example.password);
   client.connect(conOptions);
   ```

   Consulte el código en cursiva en PubSyncJAAS.Java utilizando las constantes establecidas en Example.java.

10. Establezca `Example.TCPAddress` en la dirección de socket del canal de telemetría que ha configurado para utilizar la configuración de JAAS , `MyLoginExample`. Por ejemplo, utilice 1884 como número de puerto.
11. Ejecute PubSyncJAAS varias veces para ver el inicio de sesión del cliente y que se acepte o se rechace.

Se genera una excepción cada vez que se rechaza el intento de inicio de sesión.

Resultados

Figura 103 en la página 510 muestra los resultados de la ejecución de JAAS PubSyncJAAS.Java dos veces. Los registros de anotaciones se muestran en Figura 104 en la página 510.

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:05 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_61c57a18_4bf7_40d" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Client exception caught
Client is not connected (32104)
    at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:88)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:105)
    at com.ibm.micro.client.mqttv3.MqttTopic.publish(MqttTopic.java:68)
    at com.ibm.mq.id.PubSync.main(PubSync.java:24)
```

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:40 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_1d1599a0_50f5_4ea" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Delivery token "1731749688" has been received: true
```

Figura 103. Salida de consola de PubSyncJAAS.java

El archivo de registro MyLogin.log se almacena en *WMQ Data directory*; por ejemplo, C:\IBM\MQ\Data\MyLogin.log:

```
Called JAASLogin.login at Hello World Fri Jun 04 08:31:05 BST 2010
Username: "Admin", Password: "Password" loggedIn: false
Called JAASLogin.login at Hello World Fri Jun 04 08:31:40 BST 2010
Username: "Admin", Password: "Password" loggedIn: true
```

Figura 104. MyLogin.log

Ejemplos

El código en cursiva en Figura 105 en la página 511 es la modificación del JAASLoginModule.javade ejemplo.

```
package security.jaas;
import java.io.FileWriter;
import java.io.PrintWriter;

public class JAASLogin implements javax.security.auth.spi.LoginModule {
  private javax.security.auth.Subject                  subject;
  private javax.security.auth.callback.CallbackHandler callbackHandler;
  JAASPrincipal                                        principal;
  boolean                                              loggedIn = false;
  public void initialize(javax.security.auth.Subject subject,
      javax.security.auth.callback.CallbackHandler callbackHandler,
      java.util.Map<String, ?> sharedState, java.util.Map<String, ?> options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
  }
  public boolean login() throws javax.security.auth.login.LoginException {
    try {
      javax.security.auth.callback.Callback[] callbacks = new
javax.security.auth.callback.Callback[2];
      callbacks[0] = new javax.security.auth.callback.NameCallback(
          "NameCallback");
      callbacks[1] = new javax.security.auth.callback.PasswordCallback(
          "PasswordCallback", false);

      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // login half the users randomly
      PrintWriter pw = new PrintWriter(new FileWriter(System
          .getProperty("user.dir")
          + "\\mylogin.log", true));
      pw.println("Called JAASLogin.login at "
          + System.getProperty("publication", "Hello World "
              + String.format("%tc", System.currentTimeMillis())));
      if (Math.random() < 0.5)
        loggedIn = true;
      pw.println("Username: \"" + username + "\", Password: \""
          + String.valueOf(password) + "\" loggedIn: " + loggedIn);
      pw.close();
      if (!loggedIn)
        throw new javax.security.auth.login.FailedLoginException("Login failed");
      principal = new JAASPrincipal(username);
    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }
    return loggedIn;
  }
  public boolean abort() throws javax.security.auth.login.LoginException {
    logout();
    return true;
  }
  public boolean commit() throws javax.security.auth.login.LoginException {
    if (loggedIn) {
      if (!subject.getPrincipals().contains(principal))
        subject.getPrincipals().add(principal);
    }
    return true;
  }
  public boolean logout() throws javax.security.auth.login.LoginException {
    subject.getPrincipals().remove(principal);
    principal = null;
    loggedIn = false;
    return true;
  }
}
```

Figura 105. MyLogin.java

Figura 106 en la página 512 es el código de ejemplo JAASLoginPrincipal.java, copiado en el paquete security.jaas. La finalidad de JAASLoginPrincipal es implementar la interfaz

`java.security.Principal` para mantener un registro de los usuarios que han iniciado correctamente la sesión en `MyLogin`.

```
package security.jaas;
public class JAASPrincipal implements java.security.Principal,
    java.io.Serializable {
  private static final long serialVersionUID = 1L;
  String                    name;
  public JAASPrincipal(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return (name);
  }
  public boolean equals(Object object) {
    if (object != null && object instanceof JAASPrincipal
        && name.equals(((JAASPrincipal) object).getName()))
      return true;
    else
      return false;
  }
  public int hashCode() {
    return name.hashCode();
  }
}
```

Figura 106. JAASLoginPrincipal.java

El código en PubSync.java que se modifica para añadir un `nombre_usuario` y una `contraseña` se pone en cursiva en Figura 107 en la página 512.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setUserName(Example.username);
      conOptions.setPassword(Example.password);
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("With username \"" + conOptions.getUserName()
              + "\" and password \"" + String.valueOf(conOptions.getPassword()) + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Figura 107. PubSyncJAAS.Java

Modifique las constantes en Example.java para que coincidan con la configuración. Ignore los valores SSL para este ejemplo.

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figura 108. Example.java

Autenticación de una conexión de telemetría SSL utilizando certificados autofirmados

Utilice certificados autofirmados generados utilizando **Keytool** para autenticar una conexión SSL. Tiene la opción de autenticar el canal de telemetría, o el canal de telemetría y los clientes que se adjuntan a él. Los mensajes que fluyen en la conexión están cifrados.

Antes de empezar

Realice la tarea, "Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java" en la página 485 antes de empezar, para que PubSync.java funcione con una conexión TCP/IP no segura. En esta tarea, modifique `PubSync.java` para trabajar con una conexión SSL.

Acerca de esta tarea

Los pasos de la tarea se escriben como un ejercicio de programación. Debe adaptar el procedimiento para realizar la autenticación real en un entorno de producción.

La tarea se escribe para Windows. Cambie las vías de acceso de directorio para Linux.

Procedimiento

1. Realice la tarea, “Modificación de PubSync.java para utilizar SSL” en la página 514, para modificar PubSync.java para utilizar SSL.
2. Configure el canal de telemetría y cree los almacenes de claves para utilizar SSL.

   Autentique sólo el canal de telemetría, o el canal y los clientes que se conectan a él:

   - Realice la tarea, “Autenticación del canal de telemetría” en la página 515, para conectarse con SSL, autenticando el canal de telemetría.
   - Realice la tarea, “Autenticación del canal de telemetría y los clientes” en la página 516, para conectarse con SSL, autenticando el canal de telemetría y los clientes que se conectan a él.
3. Detenga y reinicie el servicio de telemetría (MQXR) para recoger los cambios en las configuraciones de canal de telemetría.
4. Ejecute el programa cliente para ver si la configuración funciona.

Modificación de PubSync.java para utilizar SSL

Modifique el primer ejemplo de programa de publicación para conectarse a un canal de telemetría utilizando SSL. Establezca las propiedades SSL utilizadas por el programa modificado.

Antes de empezar

Se supone que ha instalado los archivos jar del cliente MQTT v3 , Javadoc, Eclipse, ha configurado canales de telemetría y ha codificado y ejecutado PubSync.java antes de realizar esta tarea. Tiene un espacio de trabajo de Eclipse que incluye una versión en ejecución de PubSync.java.

Acerca de esta tarea

La tarea utiliza el cliente de publicador, PubSync.java, que ha creado en “Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java” en la página 485 como base. Sólo son necesarias pequeñas modificaciones para utilizar SSL; consulte Figura 109 en la página 515 y Figura 110 en la página 515.

Procedimiento

1. Haga una copia de `PubSync.java` en el paquete `com.ibm.mq.id` y denomine la copia `PubSyncSSL.java`.

   Consulte “Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando Java” en la página 485 para ver los pasos para crear PubSync.java en el paquete `com.ibm.mq.id` .
2. Establezca `Example.SSLAddress` en la dirección de socket del canal de telemetría que ha configurado para utilizar para la configuración SSL.
3. Cambie el parámetro de dirección de socket del constructor de cliente para utilizar `Example.SSLAddress`.

   ```
   MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
   ```
4. Establezca `MqttConnectOptions.SSLProperties` en `PubSyncSSL.javay` pase `MqttConnectOptions` como parámetro de `MqttClient.connect`.

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setSSLProperties(Example.getSSLSettings());
   client.connect(conOptions);
   ```

   Consulte el código en cursiva `PubSyncSSL.Java` utilizando las constantes establecidas en `Example.java`.

Ejemplos

Las modificaciones en PubSync.java para añadir SSL se muestran en Figura 109 en la página 515 en cursiva.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setSSLProperties(Example.getSSLSettings());
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("SSL Properties" + conOptions.getSSLProperties());
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Figura 109. PubSyncSSL.Java

Las modificaciones en Example.java se muestran en la Figura 110 en la página 515.

```
 public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");

 public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore", "C:\\Certificates\\SSClientKey.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword", "password");
    properties.setProperty("com.ibm.ssl.trustStore", "C:\\Certificates\\SSClientTrust.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword", "password");
    return properties;
```

Figura 110. Modificaciones en Example.java

Autenticación del canal de telemetría

Los clientes autentican el canal de telemetría para cifrar el contenido de los mensajes que fluyen en el canal y para asegurarse de que un cliente se conecta al canal de telemetría correcto. El servidor no autentica el cliente.

Acerca de esta tarea

Puede utilizar varios editores de almacén de claves diferentes para crear y gestionar certificados autofirmados. La tarea utiliza el mandato **keytool** de la línea de mandatos, que forma parte del JRE. Puede utilizar la herramienta de GUI **iKeyman**, que se suministra con WebSphere MQ para examinar almacenes de claves y generar claves. Inicie **iKeyman** utilizando el mandato **strmqikm**.

Procedimiento

1. Cree un canal de telemetría, SSLSSOptClients , que requiera una conexión SSL utilizando el asistente **Nuevo canal de telemetría** . El canal acepta clientes anónimos.

   Adapte la configuración de canal desde la siguiente stanza de configuración. No edite el archivo de propiedades de telemetría directamente; utilice el asistente.

   ```
   com.ibm.mq.MQXR.channel/SSLSSOptClients: \
   com.ibm.mq.MQXR.Port=8883;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Genere las claves para que el cliente autentique el canal de telemetría.

   a) Genere un par de claves autofirmado para el canal de telemetría en un almacén de claves nuevo, SSServerOptKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerOptKey.jks -storepass password -keypass password
   ```

   b) Exporte su certificado público como un archivo ASCII, utilizando la opción -rfc :

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerOptKey.jks -storepass password -rfc
   ```

   Si está ejecutando la tarea en Windows, efectúe una doble pulsación en SSServerPublic.cer para inspeccionar su contenido.

   c) Importe el certificado público en un nuevo almacén de confianza de cliente, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

   d) Cree un almacén de claves de cliente vacío, SSClientKey.jks.

   Keytool no tiene un mandato para crear un almacén de claves vacío. Tiene dos opciones:

   i) Ejecute **strmqikm**y cree un almacén de claves, SSClientKey.jks, pero no añada ninguna clave.

   ii) Realice el paso 3a en “Autenticación del canal de telemetría y los clientes” en la página 516, pero no utilice todavía las claves.

Autenticación del canal de telemetría y los clientes

Los clientes autentican el canal de telemetría y el canal de telemetría autentica a los clientes que se adjunten a él. Los mensajes que fluyen en el canal están cifrados.

Acerca de esta tarea

Puede utilizar varios editores de almacén de claves diferentes para crear y gestionar certificados autofirmados. La tarea utiliza el mandato **keytool** de la línea de mandatos, que forma parte del JRE. Puede utilizar la herramienta de GUI **iKeyman**, que se suministra con WebSphere MQ para examinar almacenes de claves y generar claves. Inicie **iKeyman** utilizando el mandato **strmqikm**.

El canal de telemetría se configura con un almacén de claves diferente al de la tarea, “Autenticación del canal de telemetría” en la página 515. Puede utilizar el mismo almacén de claves y omitir el paso “2” en la página 517 para añadir claves al almacén de claves.

Procedimiento

1. Cree un canal de telemetría, SSLSSReqClients , que requiera una conexión SSL utilizando el asistente **Nuevo canal de telemetría** . El canal sólo acepta clientes autenticados.

   Adapte la configuración de canal de la siguiente stanza de configuración:

   ```
   com.ibm.mq.MQXR.channel/SSLSSReqClients: \
   com.ibm.mq.MQXR.Port=8884;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Genere las claves para que el cliente autentique el canal de telemetría.

   a) Genere un par de claves autofirmado para el canal de telemetría en un almacén de claves nuevo, SSServerReqKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerReqKey.jks -storepass password -keypass password
   ```

   b) Exporte su certificado público como un archivo ASCII, utilizando la opción -rfc :

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerReqKey.jks -storepass password -rfc
   ```

   Si está ejecutando la tarea en Windows, efectúe una doble pulsación en SSServerPublic.cer para inspeccionar su contenido.

   c) Importe el certificado público en un nuevo almacén de confianza de cliente, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

3. Genere las claves para el canal de telemetría para autenticar un cliente.

   a) Genere un par de claves autofirmado para el cliente en un almacén de claves nuevo, SSClientKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSClientPrivate
           -dname "CN=mqttclient.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSClientKey.jks -storepass password -keypass password
   ```

   b) Exporte su certificado público como un archivo ASCII, utilizando la opción -rfc :

   ```
   Keytool -export -noprompt -alias SSClientPrivate -file SSClientPublic.cer
           -keystore SSClientKey.jks -storepass password -rfc
   ```

   Si está ejecutando la tarea en Windows, efectúe una doble pulsación en SSClientPublic.cer para inspeccionar su contenido.

   c) Importe el certificado público en el almacén de claves del servidor, SSServerReqKey.jks:

   ```
   Keytool -import -noprompt -alias SSClientPublic -file SSClientPublic.cer
           -keystore SSServerReqKey.jks -storepass password
   ```

   Los canales de telemetría utilizan el mismo almacén para las claves privadas y los certificados de confianza.

Autenticación de una conexión de telemetría SSL utilizando una cadena de certificados

Utilice certificados firmados obtenidos de una entidad emisora de certificados, o de implementar su propio procedimiento de certificación, para autenticar una conexión SSL. Tiene la opción de autenticar el

canal de telemetría, o el canal de telemetría y los clientes que se adjuntan a él. Los mensajes que fluyen en la conexión están cifrados.

Antes de empezar

Realice la tarea, "Autenticación de una conexión de telemetría SSL utilizando certificados autofirmados" en la página 513 antes de empezar, para que PubSyncSSL.Java funcione con una conexión TCP/IP segura utilizando certificados autofirmados.

Acerca de esta tarea

En esta tarea, modifique las tareas "Autenticación del canal de telemetría" en la página 515y "Autenticación del canal de telemetría y los clientes" en la página 516 en "Autenticación de una conexión de telemetría SSL utilizando certificados autofirmados" en la página 513, para trabajar con claves certificadas por una cadena de certificados.

Puede obtener los certificados para esta tarea de una entidad emisora de certificados o puede utilizar sitios web como http://www.openca.org/ para obtener certificados. Las entidades emisoras de certificados comerciales generalmente proporcionan certificados de prueba durante un corto período sin cargo alguno. Esta tarea se probó utilizando certificados obtenidos comercialmente.

Otra opción es crear su propio proceso de certificación y ejecutarlo en sus propios sistemas, utilizando herramientas de sitios web como https://www.openssl.org/.

Los almacenes de confianza JRE `cacerts` no se utilizan en esta tarea. Puede utilizar el almacén de confianza `cacerts` de JRE en el cliente en la tarea, "Autenticación del canal de telemetría" en la página 518, en lugar de utilizar el almacén de confianza especificado. La cadena de certificados puede estar firmada por una entidad emisora de certificados bien conocida que ya tiene su certificado raíz en el almacén de `cacerts` en el cliente. En este caso, no especifique un almacén de confianza en el cliente. Asegúrese de que si hay varios JRE instalados en el cliente que gestiona el almacén de `cacerts` correcto.

Procedimiento

1. Si todavía no lo ha hecho, realice la tarea, "Modificación de PubSync.java para utilizar SSL" en la página 514, para modificar PubSync.java para utilizar SSL.
2. Configure el canal de telemetría y cree los almacenes de claves para utilizar SSL.

   Autentique sólo el canal de telemetría, o el canal y los clientes que se conectan a él:

   - Realice la tarea, "Autenticación del canal de telemetría" en la página 518, para conectarse con SSL, autenticando el canal de telemetría.
   - Realice la tarea, "Autenticación del canal de telemetría y los clientes" en la página 520, para conectarse con SSL, autenticando el canal de telemetría y los clientes que se conectan a él.
3. Detenga y reinicie el servicio de telemetría (MQXR) para recoger los cambios en las configuraciones de canal de telemetría.
4. Ejecute el programa cliente para ver si la configuración funciona.

Autenticación del canal de telemetría

Los clientes autentican el canal de telemetría para cifrar el contenido de los mensajes que fluyen en el canal y para asegurarse de que un cliente se conecta al canal de telemetría correcto. El servidor no autentica el cliente.

Acerca de esta tarea

Puede utilizar varios editores de almacén de claves diferentes para crear y gestionar certificados. La tarea utiliza el mandato **keytool** de la línea de mandatos, que forma parte del JRE. Puede utilizar la herramienta de GUI **iKeyman**, que se suministra con WebSphere MQ para examinar almacenes de claves y generar claves. Inicie **iKeyman** utilizando el mandato **strmqikm**.

Procedimiento

1. Cree un canal de telemetría, `SSLCAOptClients` , que requiera una conexión SSL utilizando el asistente **Nuevo canal de telemetría** . El canal acepta clientes anónimos.

   Adapte la configuración de canal desde la siguiente stanza de configuración. No edite el archivo de propiedades de telemetría directamente; utilice el asistente.

   ```
   com.ibm.mq.MQXR.channel/SSLCAOptClients: \
   com.ibm.mq.MQXR.Port=8885;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Genere una clave firmada por CA para que el cliente autentique el canal de telemetría.

   a) Genere un par de claves autofirmado para el canal de telemetría en un almacén de claves nuevo, `SSServerOptKey.jks`:

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverOpt.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   El algoritmo de clave se establece en RSA porque algunas entidades emisoras de certificados lo requieren. El nombre común del certificado debe ser exclusivo, algunas entidades emisoras de certificados no emiten claves con nombres comunes idénticos.

   b) Crear una solicitud de firma de certificado (CSR) como un archivo ASCII

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerOptKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   c) Ejecute el software de la entidad emisora de certificados o inicie sesión en su sitio web. Pegue el contenido de `CAServerOptKey.csr` cuando se le solicite el archivo CSR.

   d) La entidad emisora de certificados devuelve uno o dos certificados y un archivo de respuestas firmado como archivos ASCII. Pegue el contenido en dos o tres archivos:

   Certificado raíz
   Pegar en `CARoot.cer`

   certificado intermedio
   Pegar en `CAInter.cer`

   Archivo de respuestas firmado por el servidor
   Pegar en `CAServerOpt.rsp`

   El almacén de certificados JRE no se utiliza en esta tarea. Si ha recibido un certificado raíz y una respuesta firmada de la CA, utilice el certificado raíz y la respuesta firmada en los pasos siguientes. Si ha recibido una raíz y un certificado intermedio, utilice el certificado intermedio y la respuesta firmada.

   e) Reciba la respuesta del servidor firmada en el almacén de claves del servidor desde el que ha emitido la solicitud de certificado.

   Al recibir la respuesta se modifica el certificado autofirmado para que esté firmado por la CA. Si mira el certificado en el almacén de claves antes y después de recibir la respuesta, el firmante cambia. Si no lo hace, la herramienta de gestión de claves notifica un error. Antes de utilizar el certificado, inspecciónelo y verifique que el firmante sea ahora la CA.

   ```
   Keytool -import -noprompt -alias CAServer -file CAServerOpt.rsp
           -keystore CAServerOptKey.jks -storepass password
   ```

   En algún software de gestión de claves, como **iKeyman**, recibirá, en lugar de importar, archivos de respuestas.

f) Importe el certificado de CA en el almacén de confianza del cliente.

Importe el certificado intermedio si ha recibido dos certificados de la CA, o el certificado raíz, si solo ha recibido un certificado.

Realice una de las siguientes acciones:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

O:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

Autenticación del canal de telemetría y los clientes

Los clientes autentican el canal de telemetría y el canal de telemetría autentica a los clientes que se adjunten a él. Los mensajes que fluyen en el canal están cifrados.

Acerca de esta tarea

Puede utilizar varios editores de almacén de claves diferentes para crear y gestionar certificados. La tarea utiliza el mandato **keytool** de la línea de mandatos, que forma parte del JRE. Puede utilizar la herramienta de GUI **iKeyman**, que se suministra con WebSphere MQ para examinar almacenes de claves y generar claves. Inicie **iKeyman** utilizando el mandato **strmqikm**.

El canal de telemetría se configura con un almacén de claves diferente al de la tarea, “Autenticación del canal de telemetría” en la página 518. Puede utilizar el mismo almacén de claves y omitir el paso “2” en la página 520 para añadir claves al almacén de claves.

Procedimiento

1. Cree un canal de telemetría, SSLCAReqClients , que requiera una conexión SSL utilizando el asistente **Nuevo canal de telemetría** . El canal sólo acepta clientes autenticados.

   Adapte la configuración de canal desde la siguiente stanza de configuración. No edite el archivo de propiedades de telemetría directamente; utilice el asistente.

   ```
   com.ibm.mq.MQXR.channel/SSLCAReqClients: \
   com.ibm.mq.MQXR.Port=8886;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Genere una clave firmada por CA para que el cliente autentique el canal de telemetría.

   a) Genere un par de claves autofirmado para el canal de telemetría en un almacén de claves nuevo, CAServerReqKey.jks:

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerReqKey.jks -storepass password -keypass password
   ```

   El algoritmo de clave se establece en RSA porque algunas entidades emisoras de certificados lo requieren. El nombre común del certificado debe ser exclusivo, algunas entidades emisoras de certificados no emiten claves con nombres comunes idénticos.

   b) Crear una solicitud de firma de certificado (CSR) como un archivo ASCII

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerReqKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerReqKey.jks -storepass password -keypass password
   ```

c) Ejecute el software de la entidad emisora de certificados o inicie sesión en su sitio web. Pegue el contenido de `CAServerReqKey.csr` cuando se le solicite el archivo CSR.

d) La entidad emisora de certificados devuelve uno o dos certificados y un archivo de respuestas firmado como archivos ASCII. Pegue el contenido en dos o tres archivos:

Certificado raíz
Pegar en `CARoot.cer`

certificado intermedio
Pegar en `CAInter.cer`

Archivo de respuestas firmado por el servidor
Pegar en `CAServerReq.rsp`

El almacén de certificados JRE no se utiliza en esta tarea. Si ha recibido un certificado raíz y una respuesta firmada de la CA, utilice el certificado raíz y la respuesta firmada en los pasos siguientes. Si ha recibido una raíz y un certificado intermedio, utilice el certificado intermedio y la respuesta firmada.

e) Reciba la respuesta del servidor firmada en el almacén de claves del servidor desde el que ha emitido la solicitud de certificado.

Al recibir la respuesta se modifica el certificado autofirmado para que esté firmado por la CA. Si mira el certificado en el almacén de claves antes y después de recibir la respuesta, el firmante cambia. Si no lo hace, y la herramienta de gestión de claves notifica el error. Antes de utilizar el certificado, inspecciónelo y verifique que el firmante sea ahora la CA.

```
Keytool -import -noprompt -alias CAServer -file CAServerReq.rsp
        -keystore CAServerReqKey.jks -storepass password
```

En algún software de gestión de claves, como **iKeyman**, recibirá, en lugar de importar, archivos de respuestas.

f) Importe el certificado de CA en el almacén de confianza del cliente.

Importe el certificado intermedio si ha recibido dos certificados de la CA, o el certificado raíz, si solo ha recibido un certificado.

Realice una de las siguientes acciones:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

O:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

3. Genere una clave firmada por CA para el canal de telemetría para autenticar clientes.

a) Genere un par de claves autofirmado para los clientes en un almacén de claves nuevo, `CAClientKey.jks`:

```
Keytool -genkey -noprompt -alias CAClientPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

El algoritmo de clave se establece en RSA porque algunas entidades emisoras de certificados lo requieren. El nombre común del certificado debe ser exclusivo, algunas entidades emisoras de certificados no emiten claves con nombres comunes idénticos.

b) Crear una solicitud de firma de certificado (CSR) como un archivo ASCII

```
Keytool -certreq -noprompt -alias CAClient -file CAClientKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

c) Ejecute el software de la entidad emisora de certificados o inicie sesión en su sitio web. Pegue el contenido de `CAClientKey.csr` cuando se le solicite el archivo CSR.

d) La entidad emisora de certificados devuelve uno o dos certificados y un archivo de respuestas firmado como archivos ASCII. Pegue el contenido en dos o tres archivos:

Certificado raíz
Pegar en `CARoot.cer`

certificado intermedio
Pegar en `CAInter.cer`

Archivo de respuestas firmado por el cliente
Pegar en `CAClient.rsp`

El almacén de certificados JRE no se utiliza en esta tarea. Si ha recibido un certificado raíz y una respuesta firmada de la CA, utilice el certificado raíz y la respuesta firmada en los pasos siguientes. Si ha recibido una raíz y un certificado intermedio, utilice el certificado intermedio y la respuesta firmada.

e) Reciba la respuesta de cliente firmada en el almacén de claves de cliente desde el que ha emitido la solicitud de certificado.

Al recibir la respuesta se modifica el certificado autofirmado para que esté firmado por la CA. Si mira el certificado en el almacén de claves antes y después de recibir la respuesta, el firmante cambia. Si no lo hace, y la herramienta de gestión de claves notifica el error. Antes de utilizar el certificado, inspecciónelo y verifique que el firmante sea ahora la CA.

```
Keytool -import -noprompt -alias CAClient -file CAClient.rsp
        -keystore CAClientKey.jks -storepass password
```

En algún software de gestión de claves, como **iKeyman**, recibirá, en lugar de importar, archivos de respuestas.

f) Importe el certificado de CA en el almacén de claves del servidor.

Importe el certificado intermedio si ha recibido dos certificados de la CA, o el certificado raíz, si solo ha recibido un certificado.

Realice una de las siguientes acciones:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAServerReqKey.jks -storepass password
```

O:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAServerReqKey.jks -storepass password
```

Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando C

Los pasos para crear una aplicación de publicación de cliente MQTT se describen en la guía de aprendizaje. Se explica cada línea de código C. Al final de la tarea, habrá creado un publicador MQTT.

Antes de empezar

La aplicación cliente desarrollada utiliza las bibliotecas cliente MQTT v3 C. La aplicación se conecta al daemon de WebSphere MQ Telemetry para que los dispositivos publiquen mensajes. Consulte Creación del primer publicador para ver un ejemplo de un cliente que se comunica con WebSphere MQ Telemetry.

Acerca de esta tarea

El ejemplo es una aplicación de publicación, `pubsync.c`. El programa `pubsync.c` publica un mensaje con la carga útil `Hello World!` en el tema `MQTT Example`y espera la confirmación de que la publicación se ha entregado al daemon.

Para mayor simplicidad, los códigos de retorno de algunas funciones utilizadas no se prueban para la finalización correcta. En el código de producción, se pueden comprobar los códigos de retorno para asegurarse de que el programa se comporta como se esperaba. Se debe realizar la acción adecuada si se produce un error inesperado.

Al configurar un suscriptor en `MQTT Example` , puede comprobar que la aplicación funciona.

Utilice el entorno de desarrollo C seleccionado para desarrollar, compilar y ejecutar el cliente. Si lo prefiere, puede copiar el código directamente de los ejemplos.

Procedimiento

1. Cree un nuevo archivo de origen vacío, `pubsync.c`
2. Cree un archivo, `settings.h`. Copie el código de la Figura 2 en el archivo.

   Todos los parámetros utilizados en el programa se definen en `settings.h`. Puede alterar temporalmente los valores cambiando los valores del archivo.
3. Los pasos siguientes explican el código. Siga los pasos o copie el código de la Figura 1 en `pubsync.c`.
4. Añada las sentencias de inclusión de archivo de cabecera para las bibliotecas estándar necesarias y los archivos `MQTTClient.h` y `settings.h` .

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "string.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. Inicie la definición de la función `main()` .

   ```
   int main(int argc, char* argv[])
   {
   ```
6. Defina las variables locales utilizadas en el programa.

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_message pubmsg;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   Nota: Las opciones de conexión son necesarias para la función `MQTTClient_connect` . `MQTTClient_connectOptions_initializer` contiene las opciones predeterminadas.
7. Cree un cliente.

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - `& client` es un puntero a un descriptor de contexto para el cliente recién creado. Cuando esta función devuelve un código de retorno 0, contiene un descriptor de contexto para el nuevo cliente. El ejemplo presupone que se ha realizado correctamente. Pruebe el código de error para la finalización correcta en el código de producción.
   - ADDRESS es el URI del puerto MQTT que el daemon supervisa para las solicitudes de conexión de cliente entrantes.
   - `CLIENTID` es el nombre utilizado para identificar el cliente en el daemon. Cada cliente activo debe tener un nombre exclusivo. Si duplica un identificador de cliente en dos clientes en ejecución, se genera una excepción en ambos clientes y un cliente termina. El daemon utiliza el nombre para

reconocer que un cliente tjhat se está reconectando después de una desconexión, consulte El identificador de cliente.

- `MQTTCLIENT_PERSISTENCE_NONE` especifica que el estado del cliente se mantiene en la memoria y se pierde si se produce una anomalía del sistema. `MQTTCLIENT_PERSISTENCE_DEFAULT` especifica la persistencia basada en el sistema de archivos, proporcionando cierta protección contra anomalías. Para aplicaciones más especializadas, puede utilizar `MQTTCLIENT_PERSISTENCE_USER`, que proporciona una interfaz para que implemente su propio mecanismo de persistencia. Para obtener más detalles, consulte la documentación de la API para `MQTTClientPersistence.h`. Si la persistencia es necesaria es una pregunta de diseño de aplicación. Para obtener más detalles, consulte Persistencia de mensajes.
- El puerto TCP/IP de daemon predeterminado para MQTT es 1883. En el ejemplo, la dirección predeterminada se establece en `tcp://localhost:1883`.
- Hasta que llame a la función `MQTTClient_connect` , no tendrá lugar ningún proceso de mensajes.

8. Conecte el cliente al daemon.

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
     printf("Failed to connect, return code %d\n", rc);
     exit(-1);
}
```

   - Se llama a la función `MQTTClient_connect` , pasando el descriptor de contexto del cliente y un puntero a las opciones de conexión como argumentos.
   - El código de retorno de la llamada `MQTTClient_connect` se prueba para asegurarse de que la solicitud de conexión es satisfactoria.
   - Si el `MQTTClient_connect` falla, el programa finaliza con un código de error de -1.
   - Después de que la aplicación se conecte, puede empezar a publicar y suscribirse.
   - Se envía un pequeño mensaje "keep-alive" cada 20 segundos para evitar que se cierre la conexión TCP/IP. Esta opción la establece `conn_opts.keepAliveInterval`.
   - La sesión se inicia sin comprobar la finalización de los mensajes en curso restantes de una conexión anterior porque `conn_opts.cleansession` se ha establecido en true. Para obtener más detalles, consulte Limpiar sesiones.
   - No se crea ningún mensaje de última voluntad y testamento para la conexión. Para obtener más detalles, consulte Última voluntad y testamento.
9. Rellene la estructura *MQTTClient_message* con los datos para definir la carga útil del mensaje y sus atributos.

```
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
```

   - PAYLOAD es nuestro contenido de mensaje.
   - El ejemplo utiliza una carga útil de serie pero las cargas útiles MQTT son matrices de bytes. La longitud de serie es necesaria para especificar el tamaño de carga útil.
   - El ejemplo publica un mensaje QoS=1 , por lo tanto, establezca el valor en consecuencia
   - El atributo retenido se establece en false (0) ya que el mensaje no debe ser retenido por el daemon. Para obtener más detalles, consulte Publicaciones retenidas.
10. Publique el mensaje.

```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
```

    - La función de publicación especifica el cliente, el tema y la carga útil que se va a enviar al daemon.
    - `TOPIC` se define en `settings.h` como `MQTT Example`.
    - La función también pasa un puntero a un *MQTTClient_deliveryToken*. Este puntero se llena con una señal que representa el mensaje cuando se devuelve la función.

- El mensaje ahora se transfiere de forma segura al cliente MQTT, pero todavía no se ha transferido al daemon. Si el mensaje tiene QoS=1 o 2, el mensaje se almacena localmente, en caso de que el cliente falle antes de que se complete la entrega.
- Esta función devuelve un código de error que puede probar para la finalización correcta en el código de producción.

11. Espere el acuse de recibo del servidor.

```
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

- El ejemplo `pubsync.c` espera un acuse de recibo del servidor, que confirma que el mensaje se ha entregado.
- Los argumentos de cliente y símbolo identifican el mensaje específico que el programa está esperando para completarse.
- `TIMEOUT` limita el tiempo que el programa espera a que el mensaje complete la entrega. La tarea Creación de un publicador asíncrono para MQ Telemetry Transport utilizando C muestra cómo recibir confirmaciones sin esperar utilizando funciones de devolución de llamada.
- Esta función devuelve un código de error que se puede probar para la finalización correcta en el código de producción.

12. Desconecte el cliente del daemon.

```
MQTTClient_disconnect(client, 10000);
```

- El cliente se desconecta del servidor y espera a que se completen las funciones de devolución de llamada (no utilizadas en este ejemplo) para los mensajes en curso.
- El segundo argumento especifica un tiempo de espera de inmovilización en milisegundos. El ejemplo espera hasta 10 segundos para finalizar cualquier otro trabajo que deba realizar antes de desconectar.
- Esta función devuelve un código de error que se debe probar para la finalización correcta en el código de producción.

13. Libere la memoria utilizada por el cliente y finalice el programa.

```
MQTTClient_destroy(&client);
}
```

Resultados

Para ver las publicaciones enviadas por este cliente, cree un suscriptor para el tema `MQTT Example`. Para obtener más detalles, consulte Creación de un suscriptor para MQ Telemetry Transport utilizando C

Ejemplo

La Figura 1 es una lista completa del código descrito en Procedimiento. El archivo `settings.h` de la Figura 2 le permite cambiar los parámetros predeterminados utilizados en `pubsync.c`.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"

int pubsync_main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID,  MQTTCLIENT_PERSISTENCE_NONE, NULL);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
          printf("Failed to connect, return code %d\n", rc);
         exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for up to %d seconds for publication of %s\n"
          "on topic %s for client with ClientID: %s\n",
          TIMEOUT/1000, PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figura 111. pubsync.c

```
#define ADDRESS   "tcp://localhost:1883"
#define CLIENTID  "ExampleClientPub"
#define TOPIC     "MQTT Example"
#define PAYLOAD   "Hello World!"
#define QOS      1
#define TIMEOUT   10000L
```

Figura 112. settings.h

Creación de un publicador asíncrono para MQ Telemetry Transport utilizando C

Los pasos para crear una aplicación de publicación asíncrona de cliente MQTT se describen en la guía de aprendizaje. Se explica cada línea de código C. Al final de la tarea, habrá creado un publicador asíncrono MQTT.

En esta tarea, siga una guía de aprendizaje para modificar la primera aplicación de publicación. Las modificaciones permiten que la aplicación envíe publicaciones sin esperar confirmaciones de entrega. Los acuses de recibo de entrega los recibe una función de devolución de llamada que crea.

Antes de empezar

La aplicación cliente desarrollada utiliza las bibliotecas cliente MQTT v3 C. La aplicación se conecta al daemon de WebSphere MQ Telemetry para que los dispositivos publiquen mensajes. Consulte Creación del primer publicador para ver un ejemplo de un cliente que se comunica con WebSphere MQ Telemetry.

Acerca de esta tarea

El ejemplo es una aplicación de publicación, `pubasync.c`. El programa `pubasync.c` publica un mensaje con la carga útil `Hello World!` en el tema `MQTT Example`, sin esperar la confirmación de que la publicación se ha entregado al daemon. Los acuses de recibo de entrega se reciben en una función de devolución de llamada, `MQTTClient_deliveryComplete`.

Para mayor simplicidad, los códigos de retorno de algunas funciones utilizadas no se prueban para la finalización correcta. En el código de producción, se pueden comprobar los códigos de retorno para

asegurarse de que el programa se comporta como se esperaba. Se debe realizar la acción adecuada si se produce un error inesperado.

Al configurar un suscriptor en `MQTT Example` , puede comprobar que la aplicación funciona.

Utilice el entorno de desarrollo C seleccionado para desarrollar, compilar y ejecutar el cliente.

Los pasos de Procedimiento modifican la aplicación `pubsync.c` desde “Creación de la primera aplicación de publicación de MQ Telemetry Transport utilizando C” en la página 522. Si lo prefiere, puede copiar el código directamente de los ejemplos.

Procedimiento

1. Cree un nuevo archivo de origen vacío, `callback.h`.
2. Copie el código de la Figura 2 en el archivo.
   - `callback.h` declara los tres métodos de devolución de llamada necesarios para la operación de cliente asíncrona.
   - También se declara una variable, *deliveredtoken*. A esto accede el programa principal y la devolución de llamada en distintas hebras de ejecución. Por lo tanto, se declara volátil. Cuando utilice devoluciones de llamada, asegúrese de que se acceda a las variables relevantes de forma segura.
3. Cree un nuevo archivo de origen vacío, `callback.c`.
4. Copie el código de la Figura 3 en el archivo.
   - `callback.c` implementa los tres métodos de devolución de llamada utilizados por el cliente para la operación asíncrona, `delivered`, `msgarrvd`y `connlost`.
5. Añada una sentencia de inclusión para `callback.h` después de las otras inclusiones en `pubasync.c`.

   ```
   #include "callback.h"
   ```

6. Copie el contenido de `pubsync.c` en un archivo nuevo, `pubasync.c`.
7. Justo antes de la llamada a la función `MQTTClient_connect` en `pubasync.c`, establezca los métodos de devolución de llamada para el cliente.

   ```
   MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
   ```

   - Debe especificar tres funciones de devolución de llamada. Estas funciones se implementan en `callback.c`.
   - Se llama a `MQTTClient_messageArrived` cuando se envía un mensaje al cliente debido a una suscripción coincidente. Esto debe devolver true cuando la aplicación cliente haya recibido correctamente el mensaje recibido. La devolución de false indica al cliente que la aplicación ha tenido un problema al recibir el mensaje.
   - Se llama a `MQTTClient_connectionLost` cuando el cliente pierde su conexión con el servidor.
   - Se llama a `MQTTClient_deliveryComplete` cuando un mensaje QoS1 o QoS2 ha llegado y ha sido reconocido por el servidor. No se llama para los mensajes QoS0 . En el ejemplo, esta función guarda la señal del mensaje entregado en *deliveredtoken* para indicar que ha llegado un mensaje.
   - Se debe llamar a `MQTTClient_setCallbacks` mientras el cliente está desconectado del servidor.
   - El segundo argumento le permite pasar información contextual a las funciones de devolución de llamada. Esto no se utiliza en el ejemplo, por lo que se establece en NULL.
8. Inmediatamente antes de la llamada a `MQTTClient_publishMessage`, borre *deliveredtoken*. `MQTTClient_deliveryComplete` para establecer *deliveredtoken* cuando se recibe una señal.

   ```
   deliveredtoken = 0;
   ```

9. Elimine la llamada `MQTTClient_waitForCompletion` y la sentencia `printf` que le siguen y sustitúyala por un bucle que espere una coincidencia de la señal original y la señal recibida en la devolución de llamada.

   ```
   while(deliveredtoken != token);
   ```

   Este es un ejemplo y no se enfrenta a una serie de situaciones que se deben acomodar en el diseño de código de producción. Estas situaciones incluyen:

   - En caso de que la entrega no se haya completado, se puede implementar un tiempo de espera
   - Pueden estar en curso varios mensajes. El programa de ejemplo sólo permite comprobar una señal de entrega a la vez.
10. Desconecte el cliente del daemon.

    ```
    MQTTClient_disconnect(client, 10000);
    ```

    - El cliente se desconecta del servidor y espera a que se completen las funciones de devolución de llamada para los mensajes en curso.
    - El segundo argumento especifica un tiempo de espera de inmovilización en milisegundos. El ejemplo espera hasta 10 segundos para finalizar cualquier otro trabajo que deba realizar antes de desconectar.
    - Esta función devuelve un código de error que se debe probar para la finalización correcta en el código de producción.
11. Libere la memoria utilizada por el cliente y finalice el programa.

    ```
    MQTTClient_destroy(&client);
    }
    ```

Resultados

Para ver la publicación enviada por este cliente, cree un suscriptor para el tema `MQTT Example` . Para obtener más detalles, consulte Creación de un suscriptor para MQ Telemetry Transport

Ejemplo

`pubasync.c`, `callbacks.c` y `callbacks.h` son listados completos del código descrito en Procedimiento.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    deliveredtoken = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for publication of %s\n"
           "on topic %s for client with ClientID: %s\n", PAYLOAD, TOPIC, CLIENTID);
    while(deliveredtoken != token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figura 113. pubasync.c

```
MQTTClient_deliveryComplete delivered;
MQTTClient_messageArrived msgarrvd;
MQTTClient_connectionLost connlost;

extern volatile MQTTClient_deliveryToken deliveredtoken;
```

Figura 114. callback.h

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt)
{
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
    }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Figura 115. callback.c

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientPub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Figura 116. settings.h

Creación de un suscriptor para MQ Telemetry Transport utilizando C

Los pasos para crear una aplicación de suscriptor de cliente MQTT se describen en la guía de aprendizaje. Se explica cada línea de código C. Al final de la tarea, habrá creado un suscriptor MQTT.

Antes de empezar

La aplicación cliente desarrollada utiliza las bibliotecas cliente MQTT v3 C. La aplicación se conecta al daemon de WebSphere MQ Telemetry para que los dispositivos publiquen mensajes. Consulte Creación del primer publicador para ver un ejemplo de un cliente que se comunica con WebSphere MQ Telemetry.

Acerca de esta tarea

El ejemplo es una aplicación de suscriptor, `subscribe.c`. El programa `subscribe.c` se suscribe al tema `MQTT Example` y espera a que las publicaciones coincidan con la suscripción hasta que el usuario finalice el programa.

Un suscriptor crea una suscripción a un tema y espera mensajes que coincidan con el tema de suscripción. Los mensajes publicados mientras el cliente está desconectado y que coinciden con una suscripción creada anteriormente por el cliente, se pueden recibir cuando el cliente se vuelva a conectar. El daemon o servicio de telemetría WebSphere MQ (MQXR) para dispositivos reconoce un cliente que se ha conectado previamente mediante el identificador de cliente. Para obtener más información, consulte El identificador de cliente. El atributo booleano `MQTTClient_connectOptions.cleansession`

controla si las publicaciones enviadas anteriormente se reciben o no. Para obtener más información, consulte “Limpiar sesiones” en la página 539.

Para mayor simplicidad, los códigos de retorno de algunas funciones utilizadas no se prueban para la finalización correcta. En el código de producción, se pueden comprobar los códigos de retorno para asegurarse de que el programa se comporta como se esperaba. Se puede realizar la acción adecuada si se produce un error inesperado.

Puede utilizar los programas de ejemplo de publicación descritos anteriormente para enviar publicaciones coincidentes al daemon de WebSphere MQ Telemetry para dispositivos. De forma alternativa, utilice el explorador de WebSphere MQ para crear publicaciones de prueba en el tema `MQTT Example` si desea conectar el cliente a un canal de WebSphere MQ Telemetry .

Las instrucciones de Procedimiento presuponen que ya ha creado `callback.c, callback.h` y los archivos `settings.h` en una de las tareas anteriores.

Utilice el entorno de desarrollo C seleccionado para desarrollar, compilar y ejecutar el cliente. Si lo prefiere, puede copiar el código directamente de los ejemplos.

Procedimiento

1. Cree una copia de `settings.h` para este ejemplo y cambie la sentencia de definición *CLIENTID* por la siguiente:

   ```
   #define CLIENTID "ExampleClientSub"
   ```

   - Si dos clientes con el mismo ID intentan conectarse a un único servidor, uno de ellos se desconecta forzosamente. Normalmente, el nuevo intento de conexión es satisfactorio y la conexión más antigua se desconecta.
   - El cambio del `ClientID` le permite utilizar los ejemplos de publicación desarrollados anteriormente para enviar mensajes a este suscriptor.
2. Cree un nuevo archivo de origen vacío, `subscribe.c`.
3. Los pasos siguientes explican el código. Siga los pasos o copie el código de Figura 117 en la página 534 en el archivo `subscribe.c`.
4. Añada las sentencias de inclusión de archivo de cabecera para las bibliotecas estándar necesarias y los archivos `MQTTClient.h` y `settings.h` .

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. Inicie la definición de la función `main()` .

   ```
   int main(int argc, char* argv[]) {
   ```
6. Defina las variables locales utilizadas en el programa.

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   Las opciones de conexión son necesarias para la función `MQTTClient_connect` . `MQTTClient_connectOptions_initializer` contiene las opciones predeterminadas.
7. Cree un cliente.

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - `& client` es un puntero a un descriptor de contexto para el cliente recién creado. Cuando esta función devuelve un código de retorno 0, el puntero contiene un descriptor de contexto para el nuevo cliente. El ejemplo presupone que se ha realizado correctamente. El código de error se puede probar para la finalización correcta en el código de producción.

- ADDRESS es el URI del puerto MQTT que el daemon supervisa para las solicitudes de conexión de cliente entrantes.
- `CLIENTID` es el nombre utilizado para identificar el cliente en el daemon. Cada cliente activo debe tener un nombre exclusivo. Si duplica un identificador de cliente en dos clientes en ejecución, se genera una excepción en ambos clientes y un cliente termina. El daemon utiliza el nombre para reconocer que un cliente se está reconectando después de una desconexión, consulte El identificador de cliente.
- `MQTTCLIENT_PERSISTENCE_NONE` especifica que el estado del cliente se mantiene en la memoria y se pierde si se produce una anomalía del sistema. `MQTTCLIENT_PERSISTENCE#_DEFAULT` especifica la persistencia basada en el sistema de archivos, proporcionando cierta protección contra anomalías. Para aplicaciones más especializadas, puede utilizar `MQTTCLIENT_PERSISTENCE_USER`, que proporciona una interfaz para que implemente su propio mecanismo de persistencia. Si la persistencia es necesaria es una pregunta de diseño de aplicación. Para obtener más detalles, consulte Persistencia de mensajes.
- El puerto TCP/IP de daemon predeterminado para MQTT es 1883. En el ejemplo, la dirección predeterminada se establece en `tcp://localhost:1883`.
- Hasta que llame a la función `MQTTClient_connect` , no tendrá lugar ningún proceso de mensajes.

8. Conectar el cliente al daemon

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
    printf("Failed to connect, return code %d\n", rc);
    exit(-1);
}
```

   - Se llama a la función `MQTTClient_connect` , pasando el descriptor de contexto del cliente y un puntero a las opciones de conexión como argumentos.
   - El código de retorno de la llamada `MQTTClient_connect` se prueba para asegurarse de que la solicitud de conexión es satisfactoria.
   - Si la llamada de conexión falla, el programa finaliza con un código de error de -1.
   - Después de que la aplicación se conecte, puede empezar a publicar y suscribirse.
   - Se envía un pequeño mensaje "keep-alive" cada 20 segundos para evitar que se cierre la conexión TCP/IP. Esta opción la establece `conn_opts.keepAliveInterval`.
   - La sesión se inicia sin comprobar la finalización de los mensajes en curso restantes de una conexión anterior porque *conn_opts.cleansession* se ha establecido en true. Para obtener más detalles, consulte Limpiar sesiones.
   - No se crea ningún mensaje de última voluntad y testamento para la conexión. Para obtener más detalles, consulte Última voluntad y testamento

9. Suscríbase al tema.

```
MQTTClient_subscribe(client, TOPIC, QOS);
```

   - Utilice la función `MQTTClient_subscribe` para suscribir la aplicación cliente al tema seleccionado. El nombre de tema puede incluir caracteres comodín. Para obtener más información, consulte “Series de tema y filtros de tema en clientes MQTT” en la página 554.
   - El valor QoS determina la calidad máxima de servicio que se aplica a los mensajes enviados a este suscriptor. El servidor envía mensajes con el valor inferior de este valor y el valor QoS para el mensaje original.
   - Esta función devuelve un código de error que se puede probar para la finalización correcta en el código de producción.

10. Espere en un bucle hasta que el usuario entre un carácter 'Q' desde el teclado.

```
do {
    ch = getchar();
    } while(ch!='Q' && ch != 'q');
```

El programa ahora espera a que lleguen los mensajes. En este ejemplo, todo el manejo de mensajes tiene lugar en la función de devolución de llamada `MQTTClient_messageArrived`. Para obtener más información, consulte “Recepción de mensajes” en la página 533.

11. Desconecte el cliente del daemon.

```
MQTTClient_disconnect(client, 10000);
```

- El cliente se desconecta del servidor y espera a que se completen las funciones de devolución de llamada (no utilizadas en este ejemplo) para los mensajes en curso.
- El segundo argumento especifica un tiempo de espera de inmovilización en milisegundos. El ejemplo espera hasta 10 segundos para finalizar cualquier otro trabajo que deba realizar antes de desconectarse.
- Esta función devuelve un código de error que se puede probar para la finalización correcta en el código de producción.

12. Libere la memoria utilizada por el cliente y finalice el programa.

```
MQTTClient_destroy(&client);
}
```

Recepción de mensajes

Acerca de esta tarea

Cuando llegan mensajes del servidor, se inicia la función `MQTTClient_messageArrived` . Los pasos siguientes explican el código.

Procedimiento

1. Inicie la definición de la función de devolución de llamada. Esta definición debe coincidir con la plantilla de función `MQTTClient_messageArrived` .

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
```

- `context` proporciona acceso al contexto pasado a la biblioteca de cliente cuando se llamó a la función `MQTTClient_setCallbacks` . Esta función no se utiliza en el ejemplo.
- `topicName` es un puntero al tema en el que se publica el mensaje recibido. Si se ha suscrito utilizando caracteres comodín, este parámetro identifica el tema específico utilizado para el mensaje.
- `topicLen` es la longitud de la serie de tema. Esta opción se proporciona para los usuarios que deben incluir caracteres NULL en las series de tema.
- `message` es un puntero a la estructura `MQTTClient_message` que contiene la carga útil del mensaje y los atributos.

2. Defina las variables locales utilizadas.

```
int i;
char* payloadptr;
```

Estas variables se utilizan en el ejemplo para imprimir la carga útil iterando sobre ella.

3. Imprimir un mensaje, mostrando el tema y la carga útil del mensaje

```
printf("Message arrived\n");
printf("     topic: %s\n",topicName);
printf("   message: ");
payloadptr = message->payload;
for(i=0; i<message->payloadlen; i++){
    putchar(*payloadptr++);
    }
putchar('\n');
```

- El ejemplo presupone que la carga útil recibida es una secuencia de caracteres imprimibles.
- Una carga útil MQTT es una matriz de bytes. La aplicación es responsable de interpretar su significado.

4. Libere la memoria utilizada para almacenar el mensaje.

```
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
```

   - En el ejemplo, todo el manejo de mensajes tiene lugar en la función de devolución de llamada.
   - Asegúrese de que las funciones de devolución de llamada son cortas y devuelva el control a su hebra de llamada lo antes posible.
   - El puntero de mensaje se pasa para su manejo en la parte principal del programa.
   - El programa principal debe liberar la memoria utilizada por el mensaje cuando se complete el proceso. `MQTTClient_freeMessage()` es una función de conveniencia que devuelve los dos bloques de memoria utilizados para mantener la estructura *MQTTClient_message* y la carga útil del mensaje en el sistema. La memoria asignada al `topicName` debe liberarse por separado, tal como se muestra.

5. Devuelve un valor true cuando la devolución de llamada ha manejado correctamente el mensaje

```
    return 1;
}
```

   - La devolución de un valor true indica que la biblioteca de cliente puede tratar el mensaje como si se entregara correctamente.
   - Si la función de devolución de llamada no puede procesar correctamente el mensaje, se devuelve un valor falso. Por ejemplo, si la devolución de llamada está colocando mensajes en una cola para que el programa principal los procese y la cola está llena, la devolución de false sería adecuada.
   - Para los mensajes QoS1 y QoS2 , la devolución de un valor falso indica que el mensaje no se ha entregado y se realizan más intentos para entregarlo.

Código de ejemplo

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc;
    int ch;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
        }
    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

    MQTTClient_subscribe(client, TOPIC, QOS);
    do {
        ch = getchar();
        } while(ch!='Q' && ch != 'q');
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figura 117. `subscriber.c`

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt) {
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
        }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
void connlost(void *context, char *cause) {
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Figura 118. `callback.h`

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientSub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Figura 119. `settings.h`

Conceptos de programación de cliente MQTT

Los conceptos descritos en esta sección le ayudan a comprender las bibliotecas de cliente Java, JavaScript y C para la versión 3.1 de MQTT protocol. Estos conceptos complementan la documentación de API que acompaña a las bibliotecas de cliente.

`com.ibm.micro.client.mqttv3` contiene las clases que proporcionan los métodos públicos para las bibliotecas de cliente para el protocolo MQTT versión 3.1 . Con la instalación de IBM WebSphere MQ Telemetryse proporciona una versión del paquete `com.ibm.micro.client.mqttv3` y los paquetes que lo acompañan que implementan el protocolo para Java SE y ME. Para obtener la versión más reciente de las bibliotecas de cliente de MQTT (Java, JavaScript y para ver o descargar la documentación de la API, consulte "Referencia de programación de cliente MQTT".

Para desarrollar y ejecutar un cliente MQTT debe copiar o instalar estos paquetes 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 la versión 3.1 de MQTT protocol. Puede implementar sus propios clientes en los distintos lenguajes adecuados a sus distintas plataformas de dispositivos. Consulte Formato y protocolo de 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 WebSphere 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

La interfaz de `MqttCallback` tiene tres métodos de devolución de llamada; consulte alguna implementación de ejemplo en Callback.java.

`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(MqttDeliveryToken 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 542. 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 `MqttDeliveryToken` y `MqttMessage` una vez 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 MQTT continúa procesando los acuses de recibo si el cliente se reconecta a la sesión anterior con el mismo `ClientIdentifier`; consulte "Limpiar sesiones" en la página 539. 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` volverá 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(MqttTopic topic, MqttMessage message)`

`messageArrived` se llama cuando una publicación llega al 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 dos formas sencillas para procesar una publicación en un momento dado:

1. Para pausar el cliente de aplicaciones hasta que se complete la entrega de la publicación; consulte Figura 88 en la página 490.
2. 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 70 en la página 538 describe los métodos del cliente Java de MQTT que envían 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 70. 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.	Ninguno.
`MqttDeliveryToken.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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Limpiar sesiones

El cliente MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Cuando se conecta una aplicación cliente 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 WebSphere MQ puede forzar el cierre de una sesión abierta y suprimir toda la información sobre la misma. 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, se eliminan las suscripciones antiguas del clientes 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 cree el cliente se añaden a todas las suscripciones que ya existían del mismo antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otra forma de entender la forma en que el atributo `cleanSession` afecta a las suscripciones es considerarlo 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.

La modalidad `cleanSession` debe definirse antes de la conexión y dura la sesión completa. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de `cleanSession=false` a `cleanSession=true`, se descartan todas las suscripciones previas del cliente y cualquier publicación 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

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere 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 contar con un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con el identificador que se haya elegido para el mismo.

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. Supongamos por ejemplo, que un dispositivo no ha funcionado bien y se lo notifica al usuario un cliente que llama al centro de atención al cliente. ¿Cómo identifica el cliente el dispositivo y cómo correlaciona el usuario dicha identificación con el servidor que está conectado al cliente normalmente? ¿Debe consultar una base de datos en la que cada dispositivo está relacionado con un identificador de cliente y un servidor? ¿El nombre del dispositivo identifica a qué servidor está conectado? Cuando examina conexiones de clientes MQTT, todas las conexiones están etiquetadas con el identificador de cliente. ¿Necesita mirar una tabla para correlacionar 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 reemplaza un dispositivo defectuoso por uno nuevo, ¿el nuevo tiene el mismo identificador que el antiguo? ¿Asigna un identificador nuevo? Si cambia un dispositivo físico, pero mantiene el mismo identificador, las publicaciones pendientes y las suscripciones activas se transfieren automáticamente al nuevo dispositivo.

¿Cómo se garantiza la exclusividad de los identificadores de cliente? Al igual que ocurre con un sistema que genera identificadores exclusivos, el usuario debe contar con un proceso fiable para establecer el identificador en el cliente. Es posible que el dispositivo del cliente sea una "caja negra" sin interfaz de usuario. ¿Se fabrica el dispositivo con un identificador de cliente como, por ejemplo, utilizando la dirección MAC? ¿O cuenta con un proceso de configuración e instalación de software que configura el dispositivo antes de que se active?

Puede crear un identificador de cliente a partir de una dirección MAC de dispositivo de 48 bits para que el identificador siga siendo corto y exclusivo. Si el tamaño de la transmisión no es algo que resulte crítico, puede utilizar los 17 bytes restantes para hacer que la dirección sea más fácil de administrar.

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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la 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 la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones

Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT

Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT

Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere 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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un

cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Publicación de última voluntad y testamento

Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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" utilizando 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 que el mensaje sea persistente en WebSphere MQy para garantizar 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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Persistencia de mensajes en clientes MQTT

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Persistencia de mensajes en clientes MQTT

Los mensajes de publicaciones son persistentes si se envían con una calidad de servicio "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 mensaje tiene dos aspectos: cómo se transfiere el mensaje y si el mensaje se pone en cola en IBM MessageSight y IBM WebSphere MQ como 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 se especifica "como máximo una vez", `QoS=0`, el cliente rechaza el mensaje tan pronto como se publique. 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 WebSphere 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 la entrega "al menos una vez", `QoS=1`o "exactamente una vez", `QoS=2`.
2. Si un mensaje está definido con `QoS` establecido en 1 ó 2, se pone en cola en IBM MessageSight y IBM WebSphere MQ como mensaje persistente. Si el mensaje está definido con `QoS=0`, el mensaje se pone en cola en IBM MessageSight y IBM WebSphere MQ como mensaje no persistente. En IBM WebSphere 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 Cliente MQTT para Java. Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT. Puede implementar las interfaces para que se adapten a un dispositivo. El cliente 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`. El cliente también tiene una implementación predeterminada para Java ME, `MqttDefaultMIDPPersistence`.

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 mensajes persistentes utilizando la clase `MqttDefaultFilePersistence` o `MqttDefaultMIDPPersistence`.

`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 va a utilizar `MqttDefaultFilePersistence` ni `MqttDefaultMIDPPersistence`.

`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`.

Como alternativa, el cliente MQTT puede crear una instancia de `MqttDefaultFilePersistence` y colocar archivos en un directorio predeterminado. El nombre del directorio es *`client identifier`*`-tcp `*`hostname portnumber`*. `"\", "\\", "/", ":"` y `" "` se eliminan de la serie de nombre de directorio.

La vía de acceso del 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`.

`rcp.data` es una propiedad asociada a la instalación de OSGi o Eclipse Rich Client Platform (RCP).

`usr.data` es el directorio en el que se ha iniciado el mandato Java que ha iniciado la aplicación.

`MqttDefaultMIDPPersistence`
`MqttDefaultMIDPPersistence` tiene un constructor predeterminado y ningún parámetro. Utiliza el paquete `javax.microedition.rms.RecordStore` para almacenar mensajes.

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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Publicaciones

Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ 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 que permite el protocolo de MQTT 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 WebSphere 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` y `qos`, pero este contenido puede 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`.

En 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 o un IBM MessageSight es el responsable de crear las 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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Calidades de servicio proporcionadas por un cliente MQTT

Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere 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 protocolo MQTT no necesitan servidores para reenviar publicaciones con un valor de `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 consecuencia, se puede enviar al receptor el mismo elemento varias veces, y que se procese 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 acuse de recibo del mensaje `"PUBREL"`, el mensaje `"PUBREL"` se envía de nuevo hasta que se recibe aquél. El emisor suprime el mensaje que guardó 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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere MQ.

Publicaciones retenidas y clientes MQTT

Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

Utilice el método `MqttMessage.setRetained` para especificar si se retiene o no una publicación en un tema.

Para suprimir una publicación retenida en IBM WebSphere MQ, ejecute el mandato MQSC de **CLEAR TOPICSTR** .

Si crea una publicación con una carga útil nula, la publicación vacía se envía a los suscriptores. Es posible que otros intermediarios de MQTT no reenvíen una publicación vacía a los suscriptores.

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.

Cuando cree o actualice una publicación retenida, envíe la publicación con una QoS o 1 o 2. Si lo envía con una QoS de 0, IBM WebSphere MQ crea una publicación retenida no persistente. La publicación no se retiene si se detiene el gestor de colas.

Utilice publicaciones retenidas para registrar el valor más reciente de una medida. Los nuevos suscriptores al tema retenido 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.

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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere 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 WebSphere 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 WebSphere MQ puede eliminar una suscripción. Listar suscripciones utilizando WebSphere MQ Explorer, o utilizando **runmqsc** o mandatos 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, se eliminan las suscripciones antiguas del clientes 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 cree el cliente se añaden a todas las suscripciones que ya existían del mismo antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otra forma de entender la forma en que el atributo `cleanSession` afecta a las suscripciones es considerarlo 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.

La modalidad `cleanSession` debe definirse antes de la conexión y dura la sesión completa. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de `cleanSession=false` a `cleanSession=true`, se descartan todas las suscripciones previas del cliente y cualquier publicación 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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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 WebSphere 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 WebSphere MQ.

Las series de tema se utilizan para enviar publicaciones a suscriptores. Cree una serie de tema utilizando 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 IBM WebSphere 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 Cliente MQTT para Java.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de 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 WebSphere MQ por clientes de MQTT deben seguir el convenio para los nombres de gestores de colas. En concreto, las series de tema no pueden contener guiones.

2. Las longitudes máximas difieren. Las series de tema de IBM WebSphere 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 WebSphere Message Broker, un nivel de tema nulo, `'...//...'` no era válido. Los niveles de tema nulos están permitidos en IBM WebSphere MQ.

A diferencia de la publicación/suscripción en IBM WebSphere 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 WebSphere 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 WebSphere 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 Cliente MQTT para Java.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.

La sintaxis de cada tipo de filtro de temas es casi la misma. La única diferencia es la forma en que los distintos intermediarios MQTT interpretan un filtro de temas. En WebSphere Message Broker V6, sólo se podía utilizar un comodín multinivel al final de un filtro de temas. En IBM WebSphere MQ, se puede utilizar un comodín multinivel en cualquier nivel del árbol de temas; por ejemplo, `USA/#/Dutchess County`.

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 MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Identificador de cliente

Señales de entrega

Publicación de última voluntad y testamento
Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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

Publicaciones
Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Calidades de servicio proporcionadas por un cliente MQTT
Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un

cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

Publicaciones retenidas y clientes MQTT
Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

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.

Conceptos de programación de clientes C

En este tema se describen las diferencias entre el cliente C y Java para la versión 3.1 de MQ Telemetry Transport. El tema complementa los conceptos de cliente y la información de referencia de C.

El tema se organiza de la misma forma que “Conceptos de programación de cliente MQTT” en la página 535. Cada cabecera corresponde a un tema de *WebSphere(r) MQ Conceptos de programación de cliente de Telemetry Transport*. Las secciones describen las diferencias entre el cliente C y el cliente Java. No se describen pequeñas diferencias en las firmas entre los métodos Java y las funciones C.

El cliente C se utiliza con más frecuencia para implementar un adaptador ligero entre un dispositivo de telemetría y el daemon de WebSphere MQ Telemetry para dispositivos. El daemon se utiliza normalmente como concentrador de red entre dispositivos de telemetría muy ligeros y el servicio de telemetría (MQXR).

El daemon de WebSphere MQ Telemetry para dispositivos también es un cliente C y se describen las diferencias en su comportamiento con respecto al servicio de telemetría (MQXR). El daemon no proporciona una implementación de JAAS o SSL para los clientes que se conectan a él.

`mqttclient.dll` y `mqttclient.lib` son las bibliotecas de Windows de 32 bits que contienen funciones de cliente para la implementación C del protocolo MQ Telemetry Transport versión 3.1 . Las bibliotecas de Linux de 32 bits son `libmqttclient.so` y `libmqttclient.a`. Dos archivos de cabecera contienen la función y otras declaraciones necesarias para las aplicaciones cliente: `MQTTClient.h` y `MQTTClientPersistence.h`. Estos archivos se proporcionan con la instalación de WebSphere MQ Telemetry.

Para desarrollar y ejecutar un cliente de MQ Telemetry Transport, debe copiar estos archivos en el dispositivo cliente. A diferencia de los clientes de WebSphere MQ , no es necesario instalar un tiempo de ejecución de cliente independiente.

Consulte las condiciones de licencia asociadas con la característica WebSphere MQ Telemetry que rigen la conexión de clientes de MQ Telemetry Transport a WebSphere MQ y el daemon de WebSphere MQ Telemetry para dispositivos.

El cliente C es una implementación de referencia de la versión 3.1 de MQ Telemetry Transport. Puede implementar sus propios clientes en los distintos lenguajes adecuados a sus distintas plataformas de dispositivos. Consulte Formato y protocolo de MQ Telemetry Transport para obtener información detallada.

El identificador de cliente MQTT

“Identificador de cliente” en la página 541	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 contar con un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con el identificador que se haya elegido para el mismo.

- Sin diferencias.

Publicaciones

“Publicaciones” en la página 547	Las publicaciones son instancias de `MqttMessage` que están asociadas con una serie de tema. El cliente de MQTT

- No se llama a la función de devolución de llamada para publicaciones con la calidad de servicio "activar y olvidar", `QoS=0`.

Señales de entrega

“Señales de entrega” en la página 542	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.

- Una señal de entrega es un `int`. Tiene un typedef de `MQTTClient_deliveryToken`
- No se llama a la función de devolución de llamada para publicaciones con la calidad de servicio "activar y olvidar", `QoS=0`.

Publicaciones retenidas

“Publicaciones retenidas y clientes MQTT” en la página 551	Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

- Los mensajes retenidos sólo se guardan en el daemon si la persistencia está configurada; consulte Guardar mensajes y suscripciones retenidos.

  Para WebSphere MQ, la calidad de servicio afecta a si un mensaje retenido se guarda de forma permanente. Si un cliente está conectado al servicio de telemetría, los mensajes retenidos con "activar y olvidar", la calidad de servicio de `QoS=0` se descarta, si el gestor de colas concluye.

Suscripciones

“Suscripciones” en la página 552	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.

- Las suscripciones duraderas sólo se guardan en el daemon si la persistencia está configurada; consulte Guardar mensajes y suscripciones retenidas.
- Las publicaciones se pueden recibir de forma síncrona. Llame a la función `MQTTClient_receive` .

Devoluciones de llamada y sincronización

“Devolución de llamadas y sincronización en las aplicaciones cliente de MQTT” en la página 536	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`.

- La operación de sincronización en el cliente C es modal. La llamada a `MQTTClient_setCallback` coloca el cliente en modalidad asíncrona.
- En modalidad síncrona, el cliente de aplicaciones debe ceder voluntariamente el control para que el cliente MQTT pueda procesar confirmaciones y emitir pings MQTT para mantener activa la red. Control de rendimiento llamando a `MQTTClient_receive` o `MQTTClient_yield`.

Series de tema y filtros

“Series de tema y filtros de tema en clientes MQTT” en la página 554	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 WebSphere MQ.

- El daemon de WebSphere MQ Telemetry para dispositivos maneja el comodín de varios niveles *#* de forma diferente a WebSphere MQ v7. */#* debe ser los dos últimos caracteres de la serie de filtro para que *#* se comporte como un comodín. En WebSphere MQ v7, `../#/..` es un uso válido del comodín multinivel. El daemon de WebSphere MQ Telemetry para dispositivos trata el comodín de varios niveles del mismo modo que WebSphere MQ Broker v6.

Calidad de servicio

“Calidades de servicio proporcionadas por un cliente MQTT” en la página 549	Un cliente MQTT ofrece tres tipos de calidades de servicio para la entrega de publicaciones a WebSphere MQ y al cliente MQTT: "como máximo una vez", "al menos una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

- Sin diferencias.

Persistencia de los mensajes

“Persistencia de mensajes en clientes MQTT” en la página 545	Los mensajes de publicaciones son persistentes si se envían con una calidad de servicio "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.

- Debido a las diferencias de enlaces de lenguaje, establezca el mecanismo de persistencia de mensajes en el cliente C de la forma siguiente. Llame al cliente MQTT C con una de las tres opciones establecidas como cuarto parámetro en `MQTTClient_create`:

  `MQTTCLIENT_PERSISTENCE_DEFAULT`
  Persistencia basada en archivo, cuyos detalles son específicos de la plataforma de cliente.

MQTTCLIENT_PERSISTENCE_NONE
Los datos sólo se guardan en la memoria y se pierden cuando se detiene el cliente. El daemon de WebSphere MQ Telemetry para dispositivos sólo da soporte a esta opción.

MQTTCLIENT_PERSISTENCE_USER
Puede desarrollar funciones para implementar su propio mecanismo de persistencia. Pase una estructura, `MQTTClient_persistence` que contenga punteros a las funciones en la llamada `MQTTClient_create` . Consulte la información de referencia del cliente MQTT C para obtener más detalles.

Limpiar sesiones

"Limpiar sesiones" en la página 539	El cliente MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

- Sin diferencias.

Última voluntad y testamento

"Publicación de última voluntad y testamento" en la página 544	Si la conexión de un cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para que envíe 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.

- Sin diferencias.

Manejo de errores de programa

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:

- Códigos de razón para todas las demás plataformas WebSphere MQ

Diseñe sus programas para que manejen todos los códigos de retorno que pueda generar cada llamada.

Interrupciones del sistema

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:

Sistemas UNIX, Linux y Windows
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 330. En el entorno CICS , los mandatos MQCMIT y MQBACK están inhabilitados, porque puede realizar las llamadas MQPUT y MQGET dentro de las 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. Con WebSphere MQ en sistemas UNIX, Linuxy 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 más información sobre los archivos de registro en los sistemas AIX, HP-UX, Linux, Solaris y Windows , consulte Administración.

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 sistemas UNIX, Linuxy 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 mensaje) del número de veces que se produce. 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.

En sistemas WebSphere MQ para Windows, UNIX y Linux , el recuento de restituciones siempre sobrevive a los reinicios del gestor de colas.

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.

Sin embargo, el campo *Report* debe analizarse inicialmente para determinar si la aplicación que ha enviado el mensaje está interesada en ser informada de algún 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.
- Indica si el contenido de los campos *MsgId* y *CorrelId* también es necesario.

Utilice el campo *Feedback* para indicar la razón por la 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, no se le informa de ningún problema a menos que complete el campo *Report* para indicar los comentarios que necesita. 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, la cola de destino no existe. 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 11.

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 el 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 hay un programa de salida de reintento para que lo utilice (cuyo nombre se mantiene en el campo *MsgRetryExitId* del atributo de canal).

Si el campo *MsgRetryExitId* está en blanco, se utilizan los valores de los atributos *MsgRetryCount* y *MsgRetryInterval* .

Si el campo *MsgRetryExitId* no está en blanco, se ejecuta el programa de salida de 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 405.

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 11 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. Puede encontrar el nombre de la cola de mensajes no entregados abriendo el objeto del gestor de colas (utilizando la llamada MQOPEN) y consultando sobre el atributo *DeadLetterQName* (utilizando la llamada MQINQ).

Cuando el gestor de colas coloca un mensaje en esta cola, añade una cabecera al mensaje, cuyo formato describe la estructura de 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 añadir una estructura MQDLH a todos los mensajes que coloque en la cola de mensajes no entregados; sin embargo, si tiene previsto utilizar el manejador de mensajes no entregados proporcionado por determinados productos WebSphere 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 viene 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 563.

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 posible indicación 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 WebSphere MQ para 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 está contenida en el campo *Reason* de la cabecera de la 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 pueda 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 obtener información sobre las tareas de programación de WebSphere MQ Multicast como, por ejemplo, la conexión a un gestor de colas y la creación de informes de excepciones.

WebSphere MQ Multicast se ha diseñado para ser lo más transparente posible para el usuario y, sin embargo, sigue siendo compatible con las aplicaciones existentes. La definición de un objeto COMMINFO y el establecimiento de los parámetros **MCAST** y **COMMINFO** del objeto TOPIC, significa que las aplicaciones existentes de WebSphere MQ no requieren una reescritura sustancial para utilizar la multidifusión. No obstante, puede que haya que tener en cuenta alguna limitación (consulte “Multidifusión y la interfaz de cola de mensajes” en la página 564 para obtener información adicional) y algunas cuestiones de seguridad (consulte Seguridad de Multicast para obtener información adicional).

Multidifusión y la interfaz de cola de mensajes

Utilice esta información para comprender los principales conceptos de MQI y cómo se relacionan con WebSphere 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`.

WebSphere MQ Multicast da soporte al uso de 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 22.

La tabla siguiente contiene una lista de todos los conceptos principales de MQI y cómo se relacionan con Multicast:

Tabla 71. 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
Segmentation	Se rechaza	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
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
	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

Tabla 71. Conceptos de MQI y cómo se relacionan con Multicast (continuación)		
Concepto MQI	**Acción cuando se intenta mediante Multicast**	**Código de razón**
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
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 continua de igual a igual de WebSphere MQ Multicast; aunque WebSphere MQ se conecta a un gestor de colas WebSphere MQ , los mensajes no fluyen a través de ese 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 de cliente automatizada

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 WebSphere MQ Multicast.

WebSphere MQ Multicast es un protocolo compartido sin conexión 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 WebSphere MQ, esta se realiza localmente en cada cliente.

Los datos se convierten en el cliente para el tráfico de WebSphere MQ Multicast. 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 424 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 WebSphere MQ manejadores de sucesos de multidifusión e informes WebSphere MQ Excepciones de multidifusión.

WebSphere MQ Multicast ayuda con la determinación de problemas llamando al manejador de sucesos para notificar sucesos de multidifusión que se notifican utilizando el mecanismo de manejador de sucesos estándar de WebSphere MQ .

Un suceso de multidifusión individual puede hacer que se llame a más de un suceso de WebSphere MQ porque puede haber varios descriptores de conexión `MQHCONN` que utilicen el mismo transmisor o receptor de multidifusión. Sin embargo, cada excepción de multidifusión hace que solo se llame a un manejador de sucesos por cada conexión de WebSphere MQ .

La constante WebSphere MQ `MQCBDO_EVENT_CALL` permite a las aplicaciones registrar una devolución de llamada para recibir sólo WebSphere MQ sucesos, y `MQCBDO_MC_EVENT_CALL` permite a las aplicaciones registrar una devolución de llamada para recibir sólo sucesos de multidifusión. Si se utilizan ambas constantes, se recibirán ambos tipos de suceso.

Solicitud de sucesos de multidifusión

WebSphere MQ Los sucesos de multidifusión utilizan 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` para el campo `cbd.Options` , el manejador de sucesos sólo se envía WebSphere MQ sucesos de multidifusión 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

WebSphere MQ Las excepciones de multidifusión 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 72. 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
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

Tabla 72. Descripciones de código de suceso de multidifusión (continuación)

Código de suceso	Descripción	Datos adicionales
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
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	

Tabla 72. Descripciones de código de suceso de multidifusión (continuación)		
Código de suceso	**Descripción**	**Datos adicionales**
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

Utilización de .NET

WebSphere MQ classes for .NET permite a un programa escrito en la infraestructura de programación .NET conectarse a WebSphere MQ como un cliente MQI de WebSphere MQ o conectarse directamente a un servidor WebSphere MQ .

Si tiene aplicaciones que utilizan .NET Framework de Microsoft y desea aprovechar los recursos de WebSphere MQ, debe utilizar las clases WebSphere MQ para .NET.

La interfaz de WebSphere MQ .NET orientada a objetos es diferente de la interfaz MQI en que utiliza métodos de objetos en lugar de utilizar los verbos MQI.

La interfaz de programación de aplicaciones de WebSphere MQ de procedimiento se basa en verbos como los de la lista siguiente:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Todos estos verbos toman, como parámetro, un manejador para el objeto WebSphere MQ en el que van a operar. Debido a que .NET está orientado a objetos, la interfaz de programación .NET cambia esta ronda. El programa consta de un conjunto de objetos WebSphere MQ , sobre los que puede actuar llamando a métodos en dichos objetos. Puede escribir programas en cualquier idioma soportado por .NET.

Cuando se utiliza la interfaz de procedimiento, se desconecta de un gestor de colas utilizando la llamada MQDISC (*Hconn*, CompCode, Reason), donde *Hconn* es un manejador para el gestor de colas.

En la interfaz .NET, el gestor de colas está representado por un objeto de 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();
```

WebSphere MQ classes for .NET es un conjunto de clases que permiten a las aplicaciones .NET interactuar con WebSphere MQ. Representan los diversos componentes de WebSphere MQ que utiliza la aplicación, como gestores de colas, colas, canales y mensajes. Para obtener detalles de estas clases, consulte Las clases e interfaces de WebSphere MQ .NET.

Para poder compilar las aplicaciones que escriba, debe tener instalado .NET Framework. Para obtener instrucciones sobre cómo instalar las clases de WebSphere MQ para .NET y .NET Framework, consulte “Instalación de clases de WebSphere MQ para .NET” en la página 573.

Conceptos relacionados

Visión general técnica

“Opciones de conexión” en la página 572
Existen tres modalidades de conexión de clases de WebSphere MQ para .NET a un gestor de colas. Estudie qué tipo de conexión se ajusta mejor a sus requisitos.

“Utilización de clases de WebSphere MQ para .NET” en la página 583
Esta colección de temas describe cómo configurar el sistema para ejecutar los programas de ejemplo para verificar las clases de WebSphere MQ para la instalación de .NET y cómo ejecutar sus propios programas.

“Resolución de problemas de WebSphere MQ .NET” en la página 586
Si un programa no se completa correctamente, ejecute una de las aplicaciones de ejemplo y siga los consejos proporcionados en los mensajes de diagnóstico.

“Escritura y despliegue de programas WebSphere MQ .NET” en la página 587
Para utilizar WebSphere MQ classes for .NET para acceder a las colas de WebSphere MQ , puede escribir programas en cualquier idioma soportado por .NET que contengan llamadas que pongan mensajes y obtengan mensajes de las colas de WebSphere MQ .

“Canal personalizado de IBM WebSphere MQ para Microsoft Windows Communication Foundation (WCF)” en la página 606
El canal personalizado de Microsoft Windows Communication Foundation (WCF) para IBM WebSphere MQ envía y recibe mensajes entre servicios y clientes WCF.

“Decidir qué lenguaje de programación utilizar” en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

“Desarrollo de aplicaciones” en la página 7
IBM WebSphere MQ proporciona varias formas en las que puede desarrollar aplicaciones para enviar y recibir mensajes que necesita para dar soporte a los procesos de negocio. También puede desarrollar aplicaciones para gestionar los gestores de colas y recursos relacionados.

Iniciación a las clases de WebSphere MQ para .NET

WebSphere MQ classes for .NET permite a un programa escrito en la infraestructura de programación .NET conectarse a WebSphere MQ como un cliente MQI de WebSphere MQ o conectarse directamente a un servidor WebSphere MQ .

Opciones de conexión

Existen tres modalidades de conexión de clases de WebSphere MQ para .NET a un gestor de colas. Estudie qué tipo de conexión se ajusta mejor a sus requisitos.

Conexión de enlaces de cliente

Para utilizar WebSphere MQ classes for .NET como un cliente MQI de WebSphere MQ , puede instalarlo con el cliente MQI de WebSphere MQ , ya sea en la máquina del servidor WebSphere MQ o en una máquina distinta. Los enlaces de cliente pueden usar transacciones XA u otras que no lo sean

Conexión de enlaces de servidor

Cuando se utiliza en modalidad de enlaces de servidor, las clases WebSphere MQ para .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 WebSphere MQ que el uso de conexiones de red.

Para utilizar la conexión de enlaces, debe instalar las clases WebSphere MQ para .NET en el servidor WebSphere MQ .

Conexión de cliente gestionado

Una conexión realizada en esta modalidad se conecta como un cliente WebSphere MQ a un servidor WebSphere MQ que se ejecuta en la máquina local o remota.

Las clases de WebSphere MQ para .NET que se conectan en esta modalidad permanecen en código gestionado .NET y no realizan llamadas a servicios nativos. Para obtener más información sobre el 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 587.

Instalación de clases de WebSphere MQ para .NET

WebSphere MQ classes for .NET, incluidos los ejemplos, se instala con WebSphere MQ. Existe un requisito previo de Microsoft .NET Framework.

La versión más reciente de WebSphere MQ classes for .NET se instala de forma predeterminada como parte de la instalación estándar de WebSphere MQ en la característica *Java y .NET Messaging and Web Services* . Para obtener instrucciones de instalación, consulte Instalación del servidor IBM WebSphere MQ en Windows o Instalación de un cliente IBM WebSphere MQ en sistemas Windows .

En un entorno de instalación múltiple, si ha instalado previamente las clases WebSphere MQ para .NET como paquete de soporte, no puede instalar WebSphere MQ a menos que desinstale primero el paquete de soporte. La característica WebSphere MQ classes for .NET que se instala con WebSphere MQ contiene la misma funcionalidad que el paquete de soporte.

También se proporcionan aplicaciones de ejemplo, así como archivos fuente. Consulte "Aplicaciones de ejemplo" en la página 584.

Para ejecutar WebSphere MQ classes for .NET en plataformas de 32 o 64 bits, debe tener instalado Microsoft .NET Framework V2.0 o posterior.

Nota: Si Microsoft .NET Framework v2.0 o superior no está instalado antes de instalar WebSphere MQ V7.0.1, la instalación del producto WebSphere MQ continuará sin errores, pero las clases WebSphere MQ para .NET no estarán disponibles. Si .NET Framework se instala después de instalar WebSphere MQ 7.0.1, los ensamblados .NET de WebSphere deben registrarse ejecutando el script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , donde *WMQInstallDir* es el directorio donde está instalado WebSphere MQ 7.0.1 . Este script instala los ensamblajes necesarios en la memoria caché de ensamblaje global (GAC). Un conjunto de archivos `amqi*.log` que registran las acciones realizadas se crean en el directorio `%TEMP%` .

Para obtener información sobre cómo utilizar el canal personalizado WebSphere MQ para Microsoft WCF con .NET 3, consulte: "Canal personalizado de IBM WebSphere MQ para Microsoft Windows Communication Foundation (WCF)" en la página 606

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.

WebSphere MQ Las clases .NET ya dan soporte a transacciones distribuidas en conexiones en modalidad de enlaces de servidor y no gestionados. En estas modalidades, las clases WebSphere MQ .NET delegan todas sus llamadas al cliente de transacciones ampliadas C, que gestiona el proceso de transacciones en nombre de .NET.

WebSphere MQ.NET ahora dan soporte a transacciones distribuidas en modalidad gestionada donde WebSphere MQ .NET Classes utiliza el espacio de nombres System.Transactions para el soporte de transacciones distribuidas. La infraestructura System.Transactions hace que la programación transaccional sea sencilla y eficiente al dar soporte a las transacciones iniciadas en todos los gestores de recursos, incluido WebSphere MQ. La aplicación .NET de WebSphere MQ puede transferir y obtener mensajes utilizando la programación de transacciones implícita de .NET o el modelo de programación de transacciones explícita. 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.

WebSphere MQ.NET utiliza Microsoft Distributed Transaction Coordinator (MS DTC) como gestor de transacciones, que coordina y gestiona la transacción entre varios gestores de recursos. WebSphere MQ se utiliza como gestor de recursos.

WebSphere MQ.NET sigue el modelo X/Open Distributed Transaction Processing (DTP). 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 modalidad gestionada” en la página 575
- 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, WebSphere MQ .NET devuelve la solicitud de preparación retrotrayendo de forma forzada la transacción. 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 Microsoft .NET System.Transactions

Las transacciones anidadas tienen un TransactionScope dentro de otro TransactionScope

- WebSphere MQ El cliente totalmente gestionado .NET no da soporte a TransactionScope anidado
- WebSphere MQ .NET no da soporte a TransactionScope anidado

Transacciones dependientes en System.Transactions

- WebSphere MQ .NET totalmente gestionado no da soporte al recurso de transacciones dependientes proporcionado por System.Transactions.
- WebSphere MQ .NET no da soporte al recurso de transacciones dependientes proporcionado por System.Transactions.

Ejemplos de producto

Los nuevos 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 578.

Transacciones distribuidas en modalidad gestionada

WebSphere MQ Las clases .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 .NET de WebSphere MQ 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 siguiente fragmento de código describe cómo una aplicación .NET de WebSphere MQ 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 código siguiente describe cómo una aplicación .NET de WebSphere MQ transfiere 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 modo no gestionado

WebSphere MQ.NET dan soporte a conexiones no gestionadas (cliente) utilizando el cliente de transacciones ampliadas y COM + /MTS como coordinador de transacciones, utilizando el modelo de programación de transacciones implícito o explícito. En la modalidad no gestionada, las clases WebSphere MQ .NET delegan todas sus llamadas al cliente de transacciones ampliadas C que gestiona 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. WebSphere MQ Las clases .NET exponen este soporte mediante su modalidad de transporte no gestionado (cliente C). Consulte Configuración de gestores de transacciones compatibles con XA

MTS se ha desarrollado como un sistema de proceso de transacciones (TP) para proporcionar las mismas características en Windows NT que las 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.

WebSphere MQ pasa a ser compatible con MSDTC implementando una interfaz (interfaz de gestor de recursos propietario) donde gestiona la correlación de llamadas XA de DTC con llamadas WebSphere MQ(X/Open). WebSphere MQ desempeña el rol de gestor de recursos.

Cuando un componente como COM + solicita acceso a un WebSphere MQ, el COM normalmente comprueba con el objeto de contexto MTS adecuado si se necesita una transacción. Si se necesita una transacción, el COM informa al DTC e inicia automáticamente una transacción integral de WebSphere MQ 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. A continuación, el DTC emite llamadas a MQMTS que, a su vez, emite llamadas a WebSphere MQ para confirmar o retrotraer la transacción.

Escritura de una aplicación .NET de WebSphere MQ utilizando un cliente no gestionado

Para ejecutarse en el contexto de COM +, una clase .NET debe heredar de System.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 conjunto en COM + utilizando la herramienta de instalador de servicios .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. .NET Framework se encarga del registro después de que el código se ejecute 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)

Las aplicaciones C# de ejemplo de producto están disponibles en WebSphere 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 debe estar en ejecución y habilitado para transacciones XA.

Acerca de esta tarea

El ejemplo es una aplicación sencilla, `SimpleXAPut` y `SimpleXAGet`. Los programas `SimpleXAPut` y `SimpleXAGet` son aplicaciones C# disponibles en WebSphere 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 `WebSphere 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 esta sección se describe el proceso de recuperación de transacciones en WebSphere MQ .NET XA utilizando la modalidad gestionada.

Visión general

En un procesamiento de transacciones distribuidas, estas pueden completar satisfactoriamente. Sin embargo, puede haber escenarios en los que una transacción falle por muchas razones. Entre dichas razones se podrían incluir un fallo del sistema, un fallo 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

desde su biblioteca de clases System.Transactions proporciona la opción para recuperar transacciones incompletas/dudosas. Este soporte de recuperación espera del gestor de recursos que mantenga registros de las transacciones y ejecute la recuperación cuando sea necesario.

Modelo de recuperación

En el modelo de recuperación de transacciones 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 (protocolo XA de Microsoft) proporcionan las opciones para configurar el DTC para que conduzca, coordine y controle la recuperación de los mismos. 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 Microsoft Distributed Transaction coordinator (DTC):

El coordinador de transacciones distribuidas de Microsoft proporciona dos tipos de procesos 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 WebSphere MQ .NET

En esta sección se describe cómo se pueden recuperar las transacciones distribuidas con las clases .NET de WebSphere MQ .

Visión general

Para recuperar una transacción incompleta se requiere información de recuperación. Los gestores de colas deben registrar la información de recuperación de las transacciones en el almacenamiento. Las clases .NET de WebSphere MQ 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 WebSphere MQ .NET es un proceso de dos etapas.

1. Registro de la información de recuperación de transacciones.
   - 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.

2. Recuperación de transacciones utilizando una aplicación de supervisor WmqDotnetXAMonitor.
   - WmqDotnetXAMonitor es una aplicación gestionada .NET que procesa mensajes en SYSTEM.DOTNET.XARECOVERY.QUEUE y recupera transacciones incompletas

   Si el MCA es capaz de colocar el mensaje en la cola de destino, genera un informe de excepción que contiene el mensaje original y lo pone en una cola de transmisión para enviarlo a la cola de respuesta especificado 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).

SYSTEM.DOTNET.XARECOVERY.QUEUE

Esta es una cola del sistema que contiene la información de recuperación de las transacciones incompletas. Esta cola se crea cuando se crea un gestor de colas.

Nota: No debe suprimir la cola SYSTEM.DOTNET.XARECOVERY.QUEUE.

Aplicación WMQDotnetXAMonitor

La aplicación WebSphere MQ .NET XA Monitor supervisa un gestor de colas determinado y recupera las transacciones incompletas si las hay. Las siguientes son transacciones consideradas incompletas y que se han recuperado:

Transacciones incompletas

- Si la transacción se ha preparado pero no se ha completado COMMIT dentro del periodo de tiempo de espera.
- Si la transacción está preparada pero el gestor de colas de WebSphere MQ ha caído.
- Si la transacción está preparado pero, a continuación, se ha cerrado el gestor de transacciones.

  La aplicación de supervisión debe ejecutarse desde el mismo sistema en el que se ejecuta la aplicación cliente WebSphere MQ .NET. Si hay aplicaciones que se ejecutan en varios sistemas conectados al mismo gestor de colas, la aplicación del supervisor debe ejecutarse en todos los sistemas. Aunque cada máquina de cliente tiene una aplicación de supervisión en ejecución para recuperar la aplicación, cada supervisor debe poder identificar el mensaje correspondiente a la transacción que estaba coordinando el MS DTC local del supervisor, de modo que pueda recuperarla y completarla.

Casos de uso de recuperación de transacciones para WebSphere MQ .NET

A continuación se muestran los distintos escenarios de uso:

- **WebSphere MQ Application using single DTC and single Queue Manager instance:** En este escenario, cuando se conecta al gestor de colas y ejecuta Unit of Work (UoW) bajo transacción, y si la transacción falla y queda incompleta, la aplicación de supervisión recupera la transacción y la completa.

  En este escenario, habrá una única instancia de la aplicación de supervisión en ejecución, ya que un único gestor de colas está asociado a las transacciones.
- **Varias aplicaciones WebSphere MQ que utilizan una única instancia de DTC y un único gestor de colas:** en este escenario, hay más de una aplicación WMQ 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 supervisora recupera y termina dichas transacciones de todas las aplicaciones.

  En este escenario, se ejecuta una única aplicación de supervisión, ya que se utiliza un gestor de colas en las transacciones.
- **Varias aplicaciones WebSphere MQ , varias DTC, distintas instancias de gestor de colas:** en este escenario, hay más de una aplicación WMQ bajo diferentes 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 escenario, sólo habrá una instancia de la aplicación de supervisión en ejecución por cliente (usuario o sistema), ya que cada cliente tiene su propio gestor de colas utilizado en las transacciones.

- **Varias aplicaciones WebSphere MQ , varias DTC, varias instancias de gestores de colas iguales:** en este escenario, hay más de una aplicación WMQ 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 escenario, sólo habrá una única instancia de aplicación de supervisión 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 WebSphere MQ , DTC único, distintas instancias de gestor de colas:** en este escenario, hay más de una aplicación WMQ bajo un único DTC (es decir, en un sistema, hay más de una aplicación WMQ 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 escenario, habrá tantas instancias de aplicación de supervisión en ejecución como gestores de colas conectados, ya que cada aplicación tiene su propio gestor de colas utilizado en las transacciones y cada una de ellas debe recuperarse.

  Nota: Si la aplicación de supervisión no está ejecutando en segundo plano, puede inciarla.

Utilización de la aplicación WMQDotnetXAMonitor

La aplicación de supervisión XA debe ejecutarse 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 las clases .NET de WebSphere MQ .

Mandato para iniciar la aplicación de supervisión

```
WmqDotnetXAMonitor.exe -m <QueueManagerName> -n <ConnectionName> -c <Channel> -i
```

Donde

- **n** -Nombre de conexión en formato de host (puerto). El nombre de conexión 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 de supervisión ejecuta la recuperación para cada uno de los nombres de conexión especificados en la lista separada por comas.
- **c** -Nombre de canal.
- **m** -Nombre del gestor de colas. Opcional
- **i** -Finalización de ramificación heurística. Opcional

La aplicación de supervisión 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, el supervisor XA examina la cola para ver si hay mensajes y comprueba si el mensaje cumple los criterios de transacción incompleta.
3. Si alguno de los mensajes cumple los criterios de transacción incompleta, el supervisor lo extrae y recupera la información de recuperación de transacción.

4. A continuación, determina si la información de recuperación está relacionada con la DTC de MS local. Si es así, procede a la recuperación de la transacción. De lo contrario, vuelve atrás para examinar el siguiente mensaje.
5. A continuación, hace llamadas al gestor de colas para recuperar la transacción incompleta.

Valores del archivo de configuración de la aplicación WmqDotNETXAMonitor
Para supervisar la aplicación, también se pueden proporcionar entradas utilizando el archivo de configuración de la aplicación. Se proporciona un archivo de configuración de aplicación de ejemplo con WebSphere MQ .NET. Este archivo se puede modificar según sus requisitos.

El archivo de configuración de aplicación tiene la prioridad más alta cuando se consideran los valores de entrada. Si se proporcionan valores de entrada en la línea de mandatos y en el archivo de configuración de aplicación, se tienen en cuenta los valores de la configuración de aplicación.

Archivo de configuración de aplicación de ejemplo.

```
<?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>
```

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/Rollback
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Rollback
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

Utilización de clases de WebSphere MQ para .NET

Esta colección de temas describe cómo configurar el sistema para ejecutar los programas de ejemplo para verificar las clases de WebSphere MQ para la instalación de .NET y cómo ejecutar sus propios programas.

Configuración del gestor de colas para aceptar conexiones de cliente TCP/IP

Para configurar un gestor de colas para que acepte solicitudes de conexión entrantes de los clientes:

1. Defina un canal de conexión de servidor:
   a. Inicie el gestor de colas.
   b. Defina un canal de ejemplo denominado NET.CHANNEL[3]:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for WebSphere MQ classes for .NET')
```

2. Inicie un proceso de escucha:

```
runmqlsr -t tcp [-m qmnqme] [-p portnum]
```

   Nota: Los corchetes indican parámetros opcionales; *nombreqm* no es necesario para el gestor de colas predeterminado, y el número de puerto *númpuerto* no es necesario si está utilizando el valor predeterminado (1414).

Aplicaciones de ejemplo

Para ejecutar sus propias aplicaciones .NET, utilice las instrucciones para los programas de verificación, sustituyendo el nombre de aplicación en lugar de las aplicaciones de ejemplo.

Se proporcionan cinco aplicaciones de ejemplo:

- 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 idioma 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:

[3] En este ejemplo, no estamos considerando las implicaciones de seguridad. Para un sistema de producción, considere la posibilidad de utilizar SSL o una salida de seguridad. Consulte Seguridad para obtener más información.

- 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 WebSphere 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.

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 WebSphere 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

Para compilar las aplicaciones de ejemplo, se proporciona un archivo por lotes para cada lenguaje.

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 de compilación:

```
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
```

Resolución de problemas de WebSphere MQ .NET

Si un programa no se completa correctamente, ejecute una de las aplicaciones de ejemplo y siga los consejos proporcionados en los mensajes de diagnóstico.

Estas aplicaciones de ejemplo se describen en “Utilización de clases de WebSphere MQ para .NET” en la página 583.

Si los problemas continúan y necesita ponerse en contacto con el equipo de servicio de IBM , es posible que se le solicite que active el recurso de rastreo.

Rastreo de la aplicación de ejemplo

Para obtener instrucciones sobre cómo utilizar el recurso de rastreo, consulte “Rastreo de programas WebSphere MQ .NET” en la página 606.

Mensajes de error

Puede recibir los mensajes de error habituales siguientes:

Excepción no manejada de tipo 'System.IO.FileNotFoundException' en un módulo desconocido
Si este error se produce para amqmdnet.dll o amqmdxcs.dll, asegúrese de que ambos estén registrados en la 'Memoria caché de conjunto global' o cree un archivo de configuración que apunte a los ensamblados amqmdnet.dll y amqmdxcs.dll . Puede examinar y cambiar el contenido de la memoria caché de ensamblaje utilizando mscorcfg.msc, que se proporciona como parte de la infraestructura .NET.

Si .NET Framework no estaba disponible cuando se instaló WebSphere MQ , es posible que las clases no estén registradas en la memoria caché de ensamblaje global. Puede volver a ejecutar manualmente el proceso de registro utilizando el mandato

```
amqidnet -c MQ_INSTALLATION_PATH\bin\amqidotn.txt -l logfile.txt
```

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

La información sobre esta instalación se graba en el archivo de registro especificado (`logfile.txt` en este ejemplo).

Escritura y despliegue de programas WebSphere MQ .NET

Para utilizar WebSphere MQ classes for .NET para acceder a las colas de WebSphere MQ , puede escribir programas en cualquier idioma soportado por .NET que contengan llamadas que pongan mensajes y obtengan mensajes de las colas de WebSphere MQ .

La documentación de WebSphere MQ sólo contiene información sobre los lenguajes C#, C++ y Visual Basic.

Esta colección de temas proporciona información para ayudarle a escribir aplicaciones para interactuar con sistemas WebSphere MQ . Para obtener detalles de las clases individuales, consulte Las clases e interfaces de WebSphere MQ .NET.

Diferencias de conexión

La forma en que programa para WebSphere MQ .NET tiene algunas dependencias en las modalidades de conexión que desea utilizar.

Conexiones de cliente gestionado

Cuando se utilizan clases WebSphere MQ para .NET como cliente gestionado, existen varias diferencias con respecto a un cliente MQI estándar de WebSphere MQ .

Las siguientes características no están disponibles para un cliente gestionado:

- Compresión de canales
- soporte de SSL
- 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 .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 utilizar los diagnósticos de problemas de WebSphere MQ gestionados, no se da soporte a ninguno de los parámetros -i, -p, -s, -b o -c del mandato **strmqtrc** .

Una aplicación .NET gestionada que utilice transacciones XA no funcionará con un gestor de colas de z/OS . Un cliente gestionado. Net 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. Sin embargo, una aplicación .NET gestionada que utilice 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 explícitamente el tipo de conexión tal como 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. Sin embargo, mqm.dll y mqm.pdb sólo forman parte del paquete de cliente desde la versión 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, WebSphere MQ .NET lanza una excepción.

El establecimiento de NMQ_MQ_LIB en "gestionado" hace que el cliente utilice las pruebas de diagnóstico de problemas de WebSphere MQ gestionadas, la conversión de datos .NET y otras funciones gestionadas de nivel bajo WebSphere MQ .

Todos los demás valores de NMQ_MQ_LIB hacen que el proceso .NET utilice pruebas de diagnóstico de problemas y conversión de datos no gestionadas de WebSphere MQ y otras funciones de nivel bajo no gestionadas de WebSphere MQ (suponiendo que un cliente o servidor MQI de WebSphere MQ esté instalado en el sistema).

WebSphere MQ .NET elige el tipo de conexión como se indica a continuación:

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 está establecido en "gestionado", las pruebas de diagnóstico de problemas, la conversión de datos y otras funciones de bajo nivel no están totalmente gestionadas (suponiendo que un cliente o servidor MQI de WebSphere MQ esté 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 se ha instalado un servidor WebSphere MQ , se conecta como servidor.
6. Si se instala un cliente MQI de WebSphere MQ , se conecta como un cliente no XA.
7. En cualquier otro caso, se conecta como un cliente gestionado.

Archivos de configuración para clases de WebSphere MQ para .NET

Una aplicación cliente .NET puede utilizar un archivo de configuración de cliente MQI de WebSphere MQ y, si está utilizando el tipo de conexión gestionada, un archivo de configuración de aplicación .NET. Los valores contenidos en el archivo de configuración de aplicación tienen prioridad.

Archivo de configuración de cliente

Una aplicación cliente de WebSphere MQ para .NET puede utilizar un archivo de configuración de cliente de la misma forma que cualquier otro cliente MQI de WebSphere MQ . 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 Configuración de un cliente utilizando un archivo de configuración WebSphere MQ Archivo de configuración de cliente MQI.

Sólo los atributos siguientes en un archivo de configuración de cliente MQI de WebSphere MQ son relevantes para las clases WebSphere MQ para .NET. Si especifica otros atributos, esta acción no tendrá afecto.

Stanza	**Atributo**
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ServerConnectionParms
ClientExitPath	Vía de acceso predeterminada de las salidas
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
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 ejecutando con el tipo de conexión gestionada, también puede alterar temporalmente el archivo de configuración de cliente WebSphere MQ y las variables de entorno equivalentes utilizando el archivo de configuración de aplicación .NET.

Los valores del archivo de configuración de la aplicación .NET sólo se actúan cuando se ejecuta con el tipo de conexión gestionada y se ignoran para otros tipos de conexión.

El archivo de configuración de la aplicación .NET y su formato están definidos por Microsoft para uso general en la infraestructura .NET, pero los nombres, claves y valores de sección concretos mencionados en esta documentación son específicos de Websphere MQ.

El formato del archivo de configuración de aplicación .NET es una serie de *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 .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 las claves y nombres de sección del archivo de configuración de la aplicación .NET coinciden exactamente con las palabras clave para las estanzas y los atributos definidos en el archivo de configuración del cliente.

Consulte la documentación de Microsoft para obtener más información.

Ejemplo de fragmento de código

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 a WebSphere MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define a WebSphere MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it a WebSphere MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("A WebSphere MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }
```

```
    return 0;
  }//end of start
}//end of sample
```

Operaciones en gestores de colas

En esta sección se describe cómo conectarse y desconectarse de un gestor de colas utilizando las clases WebSphere MQ para .NET.

Configuración del entorno de WebSphere MQ

Antes de utilizar la conexión de cliente para conectarse a un gestor de colas, debe configurar el entorno de WebSphere MQ .

Nota: Este paso no es necesario cuando se utilizan clases WebSphere MQ para .NET en modalidad de enlaces de servidor.

La interfaz de programación .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 ClaseMQEnvironment .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 conectarse a un escucha de WebSphere MQ en el puerto 1414. Para especificar otro puerto, utilice el código:

```
MQEnvironment.Port = nnnn;
```

conectar con un gestor de colas

Ahora está preparado para conectarse a un gestor de colas creando 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 de consulta (`inq`) sobre el gestor de colas al intentar conectarse al 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 WebSphere 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 varios constructores para temas; para obtener más información, consulte Clase .NET de`MQTopic`.

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 de procedimiento, donde MQPUT y MQGET colocan y obtienen matrices de bytes, las clases WebSphere MQ para .NET colocan y obtienen 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 nueva instancia de mensaje, todos los parámetros MQMD se establecen automáticamente en sus valores predeterminados, tal como se define en Valores iniciales y declaraciones de idioma 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 de número que utilizan los métodos de lectura y escritura estableciendo la variable de miembro *encoding* .

Puede modificar el juego de caracteres para que se utilice para leer y escribir series estableciendo la variable de miembro *characterSet* .

Consulte ClaseMQMessage .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 .NET (utilizando ReadUTF()), esta es la forma más sencilla de enviar información de 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. También facilitan la comunicación entre WebSphere MQ y las aplicaciones JMS. Para obtener más información sobre las propiedades de mensaje en WebSphere 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
- Doble
- Flotante
- Int
- Int2
- Int4
- Int8
- Long
- Objeto
- Short
- Cadena

Por ejemplo, para obtener la propiedad WebSphere MQ myproperty (una serie de caracteres), utilice la llamada `message.GetStringProperty('myproperty')`. Opcionalmente, puede pasar un descriptor de propiedad, que WebSphere MQ completará.

Manejo de errores

Maneje los errores que surgen de las clases de WebSphere MQ para .NET utilizando bloques `try` y `catch` .

Los métodos de la interfaz .NET no devuelven un código de terminación ni un código de razón. En su lugar, emiten una excepción siempre que el código de terminación y el código de razón resultantes de una llamada WebSphere MQ no son ambos cero. Esto simplifica la lógica del programa para que no tenga que comprobar los códigos de retorno después de cada llamada a WebSphere 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 *nuevo* , se abre automáticamente para su consulta. Cuando se utiliza el método AccessQueue() para acceder a un objeto de cola, dicho objeto *no* se abre automáticamente para las operaciones inquire o 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 inherentemente multihebra. WebSphere MQ classes for .NET permite que un objeto de gestor de colas se comparta entre varias hebras, pero garantiza que se sincronice todo el acceso 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 WebSphere MQ .NET garantiza que, para una conexión determinada (instancia de objetoMQQueueManager), se sincronice todo el acceso al gestor de colas WebSphere MQ de destino. El comportamiento predeterminado es que una hebra que desea emitir una llamada a un gestor de colas se bloquee hasta que se completen 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).

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 definición de canal de cliente con .NET

Puede utilizar una tabla de definiciones de canal de cliente (CCDT) con las clases .NET para WebSphere MQ. 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 reside 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:

- Utilización del archivo de configuración de la aplicación .NET. En la sección CHANNELS, utilice las claves ChannelDefinitionDirectory para especificar el directorio donde reside 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 reside la tabla, y ChannelDefinitionFile para especificar el nombre de archivo.

Si la ubicación se especifica de más de una de estas formas, las variables de entorno tienen prioridad sobre el archivo de configuración de cliente y el archivo de configuración de aplicación .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.

Cómo determina una aplicación .NET qué definición de canal utilizar

En el entorno de cliente WebSphere MQ .NET, la definición de canal que se va a utilizar se puede especificar de varias formas diferentes. 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 .NET, nombre de sección CHANNELS, clave ServerConnectionParms (sólo se aplica 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 la CCDT se especifica en el archivo de configuración de la aplicación .NET (sólo se aplica a las 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 WebSphere 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, debe definir una nueva clase .NET que implemente la interfaz adecuada. Se definen tres interfaces de salida en el paquete WebSphere 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 (de *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.

En el retorno, el valor de *dataOffset* y *dataLength* se debe establecer para que apunte al desplazamiento y la longitud dentro de la matriz de bytes devuelta que las clases .NET deben utilizar. 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

V 7.5.0.6 A partir de Version 7.5.0, Fix Pack 6, el ID de usuario y la contraseña que se especifican con la aplicación cliente .NET gestionada se establece en la clase MQChannelDefinition de IBM WebSphere 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 415).

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 definición de canal de cliente con .NET" en la página 597 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 completo, incluida la especificación de espacio de nombres, de una clase .NET que implementa IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit o IBM.WMQ.MQReceiveExit (según corresponda). *Assembly_name* es la ubicación completa, incluida la extensión de archivo, del conjunto que aloja 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 es necesario, el código de cliente .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 definición de canal de cliente con .NET" en la página 597 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 de 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 cliente es un cliente WebSphere MQ .NET, puede optar por obtener una reconexión automática de cliente especificando un valor adecuado para CONNECT_OPTIONS_PROPERTY cuando utilice 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 Secure Sockets Layer (SSL)

La sección siguiente no se aplica al cliente gestionado.

Las clases de WebSphere MQ para aplicaciones cliente .NET dan soporte al cifrado SSL (Secure Sockets Layer). SSL proporciona cifrado de comunicaciones, autenticación e integridad de mensajes. Se suele utilizar para proteger las comunicaciones entre dos interlocutores cualesquiera en Internet o dentro de una intranet.

habilitar SSL

SSL sólo está soportado para conexiones de cliente. Para habilitar SSL, debe especificar la CipherSpec que se debe utilizar al comunicarse con el gestor de colas, y ésta debe coincidir con la CipherSpec establecida en el canal de destino.

Para habilitar SSL, especifique la CipherSpec utilizando la variable de miembro estático SSLCipherSpec de MQEnvironment. El ejemplo siguiente se conecta a un canal SVRCONN denominado SECURE.SVRCONN.CHANNEL, que se ha configurado para requerir SSL con una CipherSpec de NULL_MD5:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "NULL_MD5";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

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 conectarse correctamente utilizando SSL, el almacén de claves de cliente debe estar configurado con la cadena de certificados raíz de la entidad emisora de certificados desde la que se puede 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 SSL, que contiene un *Nombre distinguido* (DN).

Una aplicación cliente WebSphere MQ .NET puede utilizar este DN 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 Seguridad.

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 SSL

Las clases de WebSphere MQ para .NET pueden emitir los siguientes códigos de razón al conectarse a un gestor de colas utilizando SSL:

MQRC_SSL_NOT_ALLOWED
Se ha establecido la propiedad SSLCipherSpec, pero se ha utilizado la conexión de enlaces. Sólo la conexión de cliente da soporte a SSL.

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.

Utilización de .NET Monitor

Consulte Características que solo se pueden utilizar con la instalación primaria en Windows para obtener información importante.

.NET Monitor es una aplicación similar a un supervisor desencadenante de WebSphere MQ . Puede crear componentes .NET que se instancien siempre que se reciba un mensaje en una cola supervisada y que, a continuación, procesen dicho mensaje. El supervisor .NET se inicia mediante el mandato `runmqdnm` y se detiene mediante el mandato `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 porque .NET Monitor ya ha iniciado una transacción.

El componente recibe objetos MQQueueManager, MQQueue y MQMessage de `runmqdnm`. También puede recibir una serie de parámetro de usuario si se ha especificado una, utilizando la opción de línea de mandatos *-u* , cuando se inició `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. A continuación, el componente debe procesar el mensaje según corresponda y devolver el control al supervisor .NET.

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.

Como el componente recibe objetos MQQueueManager y MQQueue, así como el mensaje, tiene información de contexto completa para ese mensaje y, por ejemplo, puede abrir otra cola en el mismo gestor de colas sin necesidad de conectarse por separado a WebSphere MQ.

Ejemplos de fragmentos de código

Este tema contiene dos ejemplos de componentes que obtienen un mensaje del supervisor .NET y lo imprimen, uno utilizando el proceso transaccional y el otro 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;
    }

    }
}
```

Compilación de programas WebSphere MQ .NET

Mandatos de ejemplo para compilar aplicaciones .NET escritas en varios lenguajes.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Para crear una aplicación C# utilizando clases WebSphere MQ para .NET, utilice el mandato siguiente:

```
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 clases WebSphere MQ para .NET, utilice el mandato siguiente:

```
vbc /r:System.dll /r:MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Para crear una aplicación C++ gestionada utilizando clases WebSphere MQ para .NET, utilice el mandato siguiente:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Para los demás lenguajes, consulte la documentación proporcionada por el proveedor del lenguaje.

Rastreo de programas WebSphere MQ .NET

En WebSphere MQ .NET, inicie y controle el recurso de rastreo como en los programas WebSphere MQ que utilizan la MQI.

Pero los parámetros -i y -p del mandato strmqtrc, que le permiten especificar identificadores de proceso y de hebra, y procesos con nombre, no tienen ningún efecto.

Normalmente, sólo es necesario utilizar el recurso de rastreo a petición del servicio IBM .

Consulte Utilización del rastreo en Windows para obtener información sobre los mandatos de rastreo.

Canal personalizado de IBM WebSphere MQ para Microsoft Windows Communication Foundation (WCF)

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

Conceptos relacionados

“Introducción al uso del canal personalizado de WebSphere MQ para WCF con .NET 3” en la página 607
Visión general de la información disponible para los programadores que utilizan el canal personalizado WebSphere MQ para Windows Communication Foundation (WCF) con .NET 3.

"Utilización de canales personalizados de WebSphere MQ para WCF" en la página 611
Visión general de la información disponible para los programadores que utilizan canales personalizados de WebSphere MQ V7 para Windows Communication Foundation (WCF).

"Utilización de los ejemplos de WCF" en la página 628
Los ejemplos de Windows Communication Foundation (WCF) proporcionan algunos ejemplos sencillos de cómo se puede utilizar el canal personalizado de WebSphere MQ .

"Determinación de problemas en el canal personalizado WCF para WebSphere MQ" en la página 634
Puede utilizar el rastreo de WebSphere MQ para recopilar información detallada sobre lo que están haciendo las distintas partes del código de WebSphere MQ . Cuando se utiliza Windows Communication Foundation (WCF), se genera una salida de rastreo independiente para el rastreo de canal personalizado WCF integrado con el rastreo de infraestructura WCF de Microsoft .

Introducción al uso del canal personalizado de WebSphere MQ para WCF con .NET 3

Visión general de la información disponible para los programadores que utilizan el canal personalizado WebSphere MQ para Windows Communication Foundation (WCF) con .NET 3.

¿Cuál es el canal personalizado de WebSphere MQ para WCF?

El canal personalizado para WebSphere MQ es un canal de transporte que utiliza el modelo de programación unificada de Microsoft Windows Communication Foundation (WCF).

La infraestructura de Microsoft Windows Communication Foundation, introducida en Microsoft .NET 3, permite que las aplicaciones y servicios .NET se desarrollen independientemente del transporte y los protocolos utilizados para conectarlos, lo que permite que se utilicen transportes o configuraciones alternativos de acuerdo con el entorno en el que se despliega el servicio o la aplicación.

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.

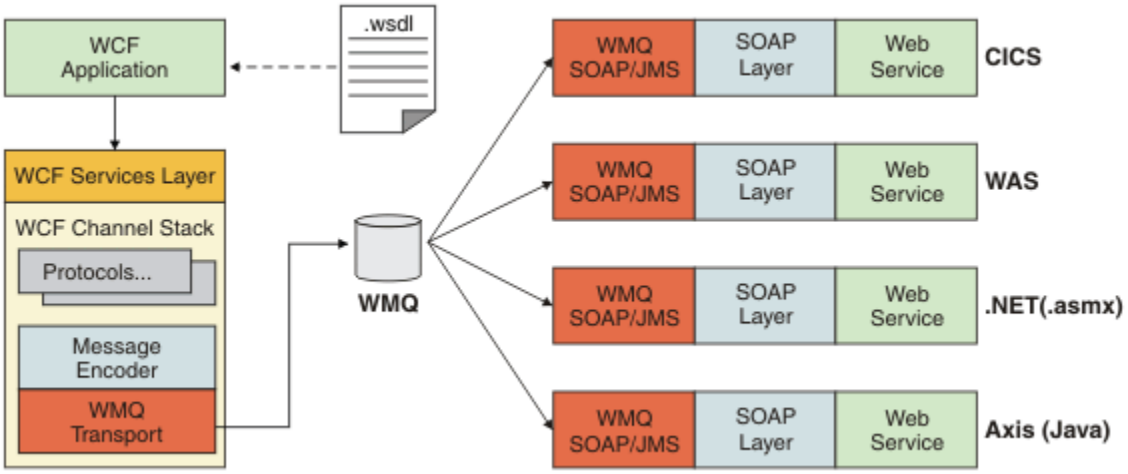
El canal personalizado para WebSphere MQ es un canal de transporte y, como tal, debe emparejarse con un codificador de mensajes y protocolos opcionales según requiera la aplicación que utiliza un enlace personalizado WCF. De esta forma, las aplicaciones que se han desarrollado para utilizar WCF pueden utilizar el canal personalizado para WebSphere 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, escalable y fiable de WebSphere MQ. Para obtener una lista completa de las funciones soportadas, consulte: "Funcionalidades y prestaciones del canal personalizado WCF" en la página 611.

¿Cuándo y por qué utilizo el canal personalizado de WebSphere MQ para WCF?

El canal personalizado WebSphere MQ se puede utilizar para enviar y recibir mensajes entre clientes y servicios WCF de la misma forma que los transportes incorporados proporcionados por Microsoft, lo que permite a las aplicaciones acceder a las características de WebSphere MQ dentro del modelo de programación unificado WCF.

Un escenario de patrón de uso típico del canal personalizado de WebSphere MQ para WCF es como una interfaz para servicios web alojados en WebSphere MQ (SOAP/JMS)

Los mensajes se transportan utilizando el formato de mensaje SOAP sobre JMS de WebSphere MQ, lo que permite a los clientes y servicios WCF llamar también o ser llamados por otras aplicaciones o entornos de alojamiento de WebSphere MQ que son compatibles con este formato, incluidos los servicios web y los clientes que se ejecutan en WebSphere Application Server, CICS, Axis v1 (Java), y .asmx (.NET), tal como se muestra en el diagrama siguiente:



Para obtener detalles sobre SOAP sobre JMS, consulte: “Transporte de WebSphere MQ para SOAP” en la página 966

Un ejemplo de un escenario típico del diagrama sería:

1. Un servicio web alojado en WebSphere Application Server y expuesto sobre WebSphere MQ utilizando el soporte para SOAP sobre JMS en WebSphere Application Server
2. La herramienta WCF puede usar el documento WSDL que describe el servicio para generar un proxy de cliente y una configuración que luego crearía la correspondiente pila de canal WCF, incluyendo el canal personalizado.
3. A continuación, la aplicación cliente puede utilizar el proxy para iniciar el servicio web de la misma forma que cualquier otro servicio web.

Normalmente, el canal se usa con un codificador de mensajes text/SOAP de WCF, pero el canal se puede emparejar con otros codificadores de mensajes WCF si es necesario. El uso de codificadores alternativos también puede proporcionar una integración limitada con aplicaciones nativas de WebSphere MQ que no dan soporte a SOAP sobre JMS, pero este no es el rol primario del canal.

Las principales ventajas de utilizar el canal personalizado en un entorno WCF son las siguientes:

- Invocación síncrona: Se soportan operaciones cliente 'transmitir y olvidar' donde el cliente está desacoplado de la disponibilidad y de las funciones del servicio como, por ejemplo, el redireccionamiento de respuestas y el salto por múltiples sitios (multi-hop).
- Características de escalado fiables: La mensajería basada en colas permite añadir capacidad a un sistema de forma predecible.
- Calidad del servicio: Los mensajes son tangibles y rastreables, y se pueden gestionar y administrar fácilmente.

Requisitos de software e instrucciones de instalación para el canal personalizado de WebSphere MQ para WCF

En este tema se describen los requisitos de software y la información de instalación para el canal personalizado de WebSphere MQ para WCF.

El canal personalizado de WebSphere MQ para WCF sólo se puede conectar a gestores de colas de WebSphere MQ V7 o superior.

Requisitos de software para el canal personalizado WCF para WebSphere MQ

Esta información lista los requisitos de software para el canal personalizado WCF para WebSphere MQ.

Entorno de ejecución

- Microsoft .NET Framework v3.0 o superior debe estar instalado en la máquina host.
- *Java y .NET Messaging and Web Services* se instala de forma predeterminada como parte del instalador de WebSphere MQ 7.0.1 . Instala los ensamblados .NET necesarios para el canal personalizado en la memoria caché de ensamblaje global.

Nota: Si Microsoft .NET Framework v2.0 o superior no está instalado antes de instalar WebSphere MQ V7.0.1, la instalación del producto WebSphere MQ continúa sin errores, pero el canal personalizado WebSphere MQ no está disponible. Si .NET Framework se instala después de instalar WebSphere MQ 7.0.1, el canal personalizado WebSphere MQ debe activarse ejecutando el script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd , donde *WMQInstallDir* es el directorio donde está instalado WebSphere MQ 7.0.1 . Este script instala los ensamblajes necesarios en la memoria caché de ensamblaje global (GAC). Un conjunto de archivos amqi*.log que registran las acciones realizadas se crean en el directorio %TEMP% . No es necesario volver a ejecutar el script amqiRegisterdotNet.cmd si .NET se actualiza a v3.0 o superior desde una versión anterior, por ejemplo, desde .NET v2.0.

Entorno de desarrollo

- Microsoft Visual Studio 2008 o Windows Software Development Kit for .NET 3.0 o posterior.
- Microsoft .NET Framework V3.5 o superior debe estar instalado en la máquina host para poder crear los archivos de solución de ejemplo.

Nota: Si Microsoft .NET Framework v2.0 o superior no está instalado antes de instalar WebSphere MQ V7.0.1, la instalación del producto WebSphere MQ continúa sin errores, pero el canal personalizado WebSphere MQ no está disponible. Si .NET Framework se instala después de instalar WebSphere MQ 7.0.1, el canal personalizado WebSphere MQ debe activarse ejecutando el script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd , donde *WMQInstallDir* es el directorio donde está instalado WebSphere MQ 7.0.1 . Este script instala los ensamblajes necesarios en la memoria caché de ensamblaje global (GAC). Un conjunto de archivos amqi*.log que registran las acciones realizadas se crean en el directorio %TEMP% . No es necesario volver a ejecutar el script amqiRegisterdotNet.cmd si .NET se actualiza a v3.0 o superior desde una versión anterior, por ejemplo, desde .NET v2.0.

Canal personalizado de WebSphere MQ para WCF: ¿Qué se ha instalado?

El canal personalizado para WebSphere MQ es un canal de transporte que utiliza el modelo de programación unificada de Microsoft Windows Communication Foundation (WCF). El canal personalizado se instala de forma predeterminada como parte de la instalación de WebSphere MQ 7.0.1 .

Canal personalizado de WebSphere MQ para WCF

El canal personalizado de WebSphere MQ para WCF se instala de forma predeterminada como parte de la instalación de WebSphere MQ 7.0.1 ; 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 a WebSphere MQ 7.0.1 desde una versión anterior, la actualización instalará el canal personalizado de WebSphere MQ para WCF de forma predeterminada si el componente Java and .NET Messaging and Web Services se ha instalado anteriormente en una instalación anterior.

El componente Java and .NET Messaging and Web Services contiene el archivo IBM.XMS.WCF.dll y el archivo IBM.XMS.WCF.dll es el conjunto de canal personalizado principal, que contiene las clases de interfaz WCF. Este archivo se instala en Global Assembly Cache (GAC) y también está disponible en el siguiente directorio: *MQ_INSTALLATION_PATH*\bin donde *MQ_INSTALLATION_PATH* es el directorio en el que está instalado WebSphere MQ 7.0.1 .

Las clases clave necesarias para utilizar el canal personalizado se encuentran en el *Espacio de nombres: IBM.XMS.WCF* y:

Nombre de enlace de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement
Importador de enlaces de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter
Config. enlaces de transporte	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig

Ejemplos de canal personalizado de WebSphere MQ

Los ejemplos proporcionan algunos ejemplos sencillos de cómo se puede utilizar el canal personalizado de WebSphere MQ para WCF. Los ejemplos y sus archivos asociados se encuentran en el directorio *MQ_INSTALLATION_PATH*`\tools\wcf\samples\` , donde *MQ_INSTALLATION_PATH* es el directorio de instalación de WebSphere MQ. Para obtener más información sobre los ejemplos de canal personalizado de WebSphere MQ , consulte: "Utilización de los ejemplos de WCF" en la página 628

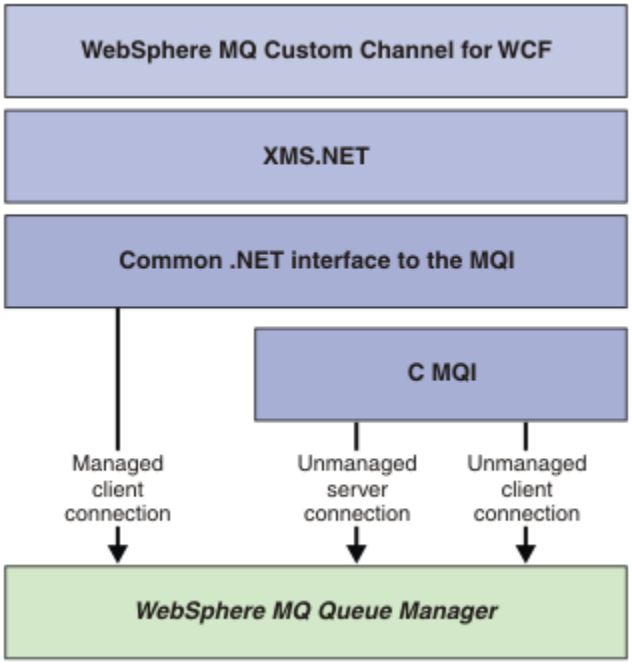
svcutil.exe.config

`svcutil.exe.config` es un ejemplo de los valores de configuración necesarios para habilitar la herramienta de generación de proxy de cliente `svcutil` de Microsoft WCF 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 WebSphere MQ. Para obtener más información sobre cómo utilizar `svcutil.exe.config`, consulte: "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 625.

Arquitectura WCF

El canal personalizado de WebSphere MQ para WCF se integra en la parte superior de la API de `IBM Message Service Client for .NET (XMS .NET)`.

La arquitectura WCF es la que se muestra en el diagrama siguiente:



Todos los componentes necesarios se instalan de forma predeterminada con la instalación de WebSphere MQ V7.0.1 .

Las tres conexiones son: Conexiones de cliente gestionadas, Conexiones de servidor no gestionadas y Conexiones de cliente no gestionadas. Para obtener más información sobre estas conexiones, consulte “Opciones de conexión WCF” en la página 615.

Utilización de canales personalizados de WebSphere MQ para WCF

Visión general de la información disponible para los programadores que utilizan canales personalizados de WebSphere MQ V7 para Windows Communication Foundation (WCF).

Microsoft Windows Communication Foundation apoya los servicios web y el soporte de mensajería en Microsoft .NET Framework 3. WebSphere MQ V7 ahora se puede utilizar como 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 SOAP sobre JMS de WebSphere MQ V7. A continuación, las aplicaciones pueden comunicarse con los servicios alojados por WCF o por la infraestructura de servicios SOAP sobre JMS de WebSphere . Para obtener detalles sobre SOAP sobre JMS, consulte: “Transporte de WebSphere MQ para SOAP” en la página 966

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

Visión general de las formas de canal personalizadas que WebSphere MQ se pueden utilizar como dentro de los canales personalizados de Microsoft Windows Communication Foundation (WCF).

El canal personalizado de WebSphere MQ para WCF da soporte a 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 WebSphere 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 WebSphere MQ para WCF se utiliza, por ejemplo, para enviar mensajes en dos direcciones de forma asíncrona; se debe utilizar la misma instancia de cliente 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

connectionFactory
El parámetro connectionFactory es obligatorio. Para ver la sintaxis de este parámetro, consulte Sintaxis de URI y parámetros para el despliegue de servicio web .

initialContextFactory
El parámetro de fábrica initialContextes necesario y se debe establecer en "com.ibm.mq.jms.Nojndi" para la compatibilidad con WebSphere Application Server y otros productos (consulte “Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP” en la página 1024).

Entrega garantizada del canal personalizado WCF

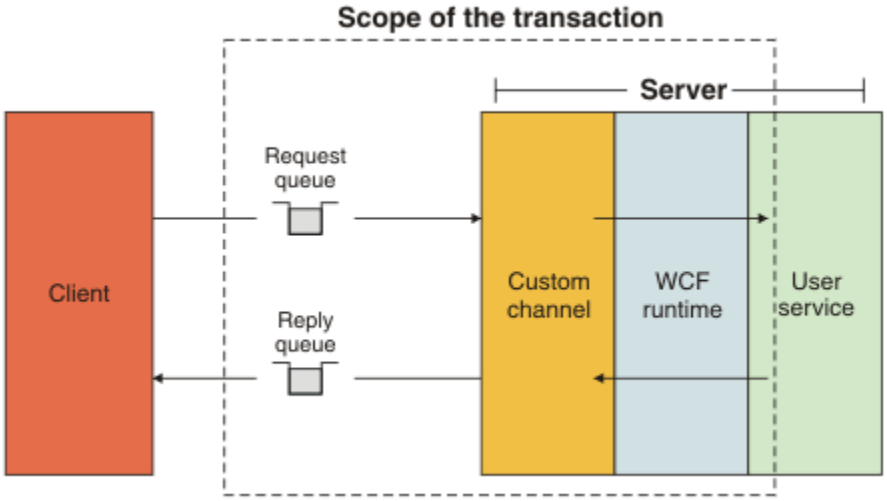
Una entrega garantizada garantiza que una solicitud o respuesta de servicio se accione y no se pierda.

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. Para obtener más información sobre cómo establecer las propiedades de un URI, consulte Sintaxis y parámetros de un URI para el despliegue de servicios web.

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 WebSphere 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 614
- 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 621
- 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 WebSphere MQ para WCF da soporte al uso de SSL sólo para conexiones de cliente no gestionadas con el gestor de colas.

SSL se puede especificar de una de estas dos maneras:

- Especifique SSL directamente en el URI de SOAP sobre JMS. Para obtener una descripción completa de las opciones SSL, consulte SSL y el transporte WebSphere MQ para SOAP
- Especifique SSL utilizando una entrada en la tabla de definiciones de canal de cliente (CCDT). Para obtener más información sobre las CCDT, consulte Tabla de definiciones de canal cliente.

Tablas de definiciones de canal de cliente (Client Channel Definition Table, CCDT) de WCF

El canal personalizado de WebSphere MQ para WCF da soporte al uso de tablas de definición 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.

No puede especificar la tabla de definiciones de canal directamente en el URI de SOAP sobre JMS. 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 definición de canal de cliente, consulte: Tabla de definición de canal de cliente .

Conceptos relacionados

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 612
- 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, las siguientes opciones de informe se establecen en los mensajes de solicitud de forma predeterminada y no se pueden configurar:

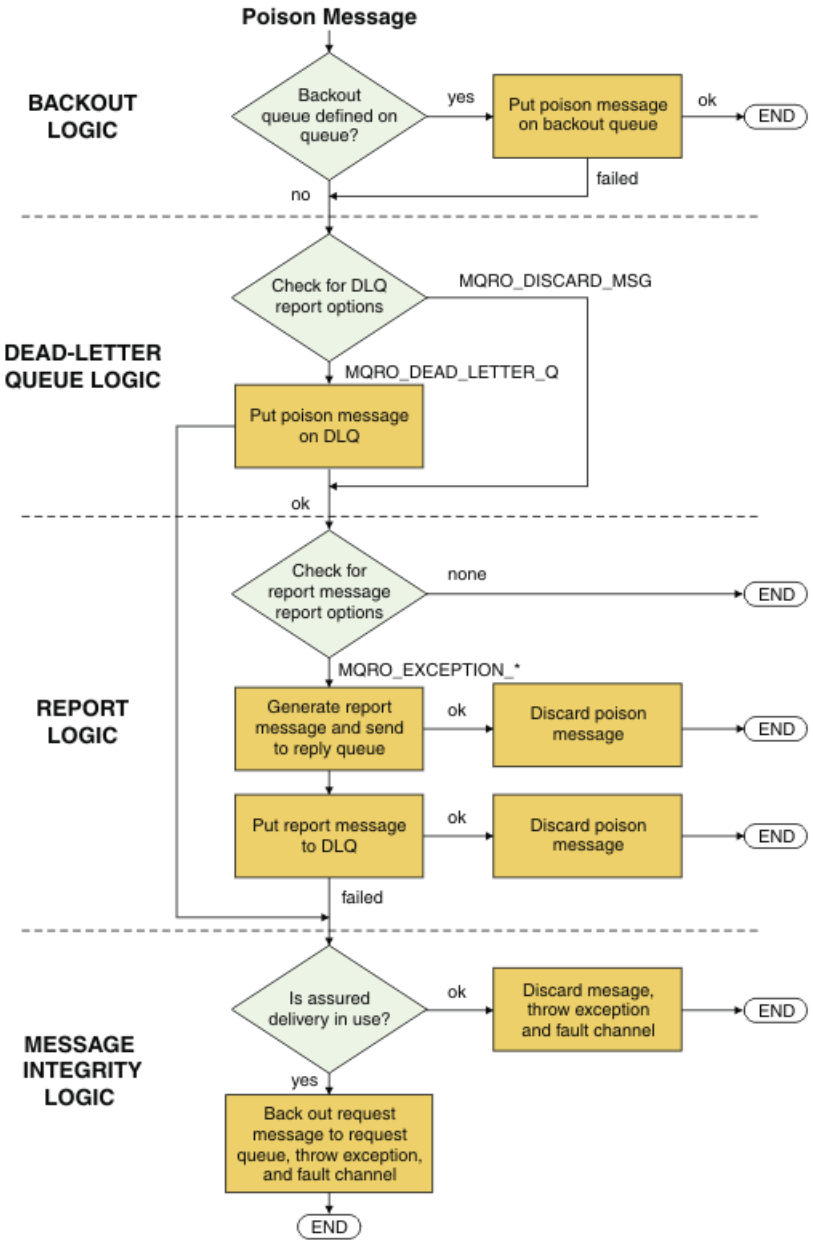
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Para SOAP sobre JMS, la siguiente opción de informe se establece en los mensajes de respuesta de forma predeterminada y no se puede configurar:

- MQRO_DEAD_LETTER_Q

Si los mensajes provienen de un origen no WCF, consulte la documentación del origen.

El siguiente diagrama muestra las posibles acciones y los pasos realizados si falla el manejo de mensajes con formato incorrecto:



Opciones de conexión WCF

Existen tres modalidades de conexión de un canal personalizado de WebSphere 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 587

Para obtener más información sobre la arquitectura WCF, consulte: “Arquitectura WCF” en la página 610

Conexión de cliente no gestionado

Una conexión realizada en esta modalidad se conecta como un cliente WebSphere MQ a un servidor WebSphere MQ que se ejecuta en la máquina local o en una máquina remota.

Para utilizar el canal personalizado WebSphere MQ para WCF como un cliente WebSphere MQ , puede instalarlo, con el cliente MQI WebSphere MQ , en el servidor WebSphere MQ o en una máquina independiente.

Conexión de servidor no gestionado

Cuando se utiliza en modalidad de enlaces de servidor, el canal personalizado de WebSphere 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 WebSphere MQ que el uso de conexiones de red.

Para utilizar la conexión de enlaces, debe instalar el canal personalizado WebSphere MQ para WCF en el servidor WebSphere MQ .

Conexión de cliente gestionado

Una conexión realizada en esta modalidad se conecta como un cliente WebSphere MQ a un servidor WebSphere MQ que se ejecuta en la máquina local o en una máquina remota.

Las clases de canal personalizadas de WebSphere MQ para .NET 3 que se conectan en esta modalidad permanecen en el código gestionado .NET y no realizan ninguna llamada a los servicios nativos. Para obtener más información sobre el 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 587.

Creación y configuración del canal personalizado de WebSphere MQ para WCF

Los canales personalizados de WebSphere MQ V7 para WCF funcionan de la misma forma que los canales WCF de transporte ofrecidos por Microsoft. El canal personalizado de WebSphere MQ para WCF se puede crear de una de dos maneras.

Acerca de esta tarea

El canal personalizado de WebSphere MQ se integra con WCF como canal de transporte WCF, y como tal debe emparejarse con un codificador de mensajes y canales de protocolo opcionales, para 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. Para el canal personalizado, el URI tiene el formato de un URI 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 WebSphere 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 WebSphere MQ para WCF, que es un canal WCF de nivel de transporte, se debe definir 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 619
- 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 la aplicación se modifica más fácilmente utilizando la herramienta del editor de configuración del servicio Microsoft denominada `SvcConfigEditor.exe` de la siguiente manera:

- 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 73. Campos de Nuevo elemento de enlace

Campo	**Valor**
Name	`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 74. Campos de Nueva configuración de enlace

Campo	**Valor**
Name	`CustomBinding_WMQ`

Tabla 74. Campos de Nueva configuración de enlace (continuación)	
Campo	**Valor**
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 617
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 621

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 617 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 **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 75. Campos de Nuevo punto final de cliente	
Campo	**Valor**
Name	`Endpoint_WMQ`
Address	*El URI de SOAP/JMS que describe los detalles de conexión WMQ necesarios para acceder al servicio.* Para obtener más detalles, consulte: "WebSphere MQ canal personalizado para el formato de dirección URI de punto final WCF" en la página 620
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 76. Campos de puntos finales de Nuevo servicio	
Campo	**Valor**
Name	`Endpoint_WMQ`

Tabla 76. Campos de puntos finales de Nuevo servicio (continuación)	
Campo	**Valor**
Address	*El URI de SOAP/JMS que describe los detalles de conexión WMQ necesarios para acceder al servicio.* Para obtener más detalles, consulte: "WebSphere MQ canal personalizado para el formato de dirección URI de punto final WCF" en la página 620
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 WebSphere 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 WebSphere MQ para WCF, que es un canal WCF de nivel de transporte, se debe definir 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: "Utilización de los ejemplos de WCF" en la página 628

Estas definiciones pueden crearse de dos formas:

- Administrativamente, proporcionando los detalles en un archivo de configuración de aplicación, tal como se describe aquí: "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 617
- Mediante programación directamente desde el código de aplicación, tal como se describe en el ejemplo siguiente.

Paso 1: Crear una instancia del elemento de enlace de transporte del canal

Añada el código siguiente a la aplicación:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

Paso 2: Establecer propiedades de enlace

Establezca las propiedades de enlace necesarias, por ejemplo, añadiendo el código siguiente a la aplicación para establecer `ClientConnectionMode`.

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

Paso 3: Crear un enlace personalizado que empareja el canal de transporte con un codificador de mensajes

Cree un enlace personalizado añadiendo el código siguiente a la aplicación:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

Paso 4: Creación del URI de SOAP/JMS

El URI de SOAP/JMS que describe los detalles de conexión de WebSphere MQ necesarios para acceder al servicio, debe proporcionarse como la dirección de punto final. Esto depende de si el canal se está utilizando para una aplicación de servicio o una aplicación cliente.

Para las aplicaciones cliente, el URI de SOAP/JMS se debe crear como `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 como se indica a continuación:

```
Uri address = new Uri("jms:/queue?
destination=SampleQ@QM1&connectionFactory=connectQueueManager(QM1)&initialContextFactory=com.ibm
.mq.jms.Nojndi");
```

Para obtener más información sobre la dirección de punto final, consulte: “WebSphere MQ canal personalizado para el formato de dirección URI de punto final WCF” en la página 620

WebSphere MQ canal personalizado para el formato de dirección URI de punto final WCF

Un URI (Universal Resource Identifier) proporciona detalles de ubicación y conexión para especificar un servicio web. Este formato de URI permite un grado completo de control sobre las opciones y parámetros específicos de SOAP/ WebSphere MQal acceder a los servicios de destino.

Un servicio web se especifica utilizando un URI (identificador universal de recursos). Esta sección especifica el formato de URI que está soportado en el transporte WebSphere MQ para SOAP. Este formato de URI permite un grado completo de control sobre las opciones y parámetros específicos de SOAP/WebSphere MQal acceder a los servicios de destino. Este formato es compatible con WebSphere Application Server (WAS) y con CICS facilitando la integración de WebSphere MQ con ambos productos.

La sintaxis del URI es la siguiente:

```
jms:/queue?name=value&name=value...
```

donde name es un nombre de parámetro y *valor* es un valor adecuado, y el elemento name=*valor* se puede repetir cualquier número de veces con la segunda y subsiguientes apariciones precedidas por un ampersand (&).

Para obtener más información sobre cómo establecer las propiedades de URI, consulte: Sintaxis de URI y parámetros para el despliegue de servicio web

Los nombres de parámetro distinguen entre mayúsculas y minúsculas, al igual que los nombres de objetos de WebSphere 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 incluyen parámetros adicionales no reconocidos, se ignoran.

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
```

Sólo se proporcionan los parámetros necesarios (`targetService` solo es necesario para los servicios .NET) y `connectionFactory` no tiene opciones.

En este ejemplo de eje, `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 eje, 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
```

Opciones de configuración de enlace WCF

En este tema se describe cómo se pueden aplicar las opciones de configuración a la información de enlace de canales personalizados y se listan las opciones disponibles.

Las opciones de configuración de enlace se pueden establecer de dos maneras distintas:

1. Administrativamente: los valores de propiedad de enlace se deben especificar en la sección de transporte de la definición de enlace personalizado en el archivo de configuración de 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 propiedad de enlace también se pueden especificar en el archivo de configuración de la aplicación, por ejemplo: `app.config`. El archivo de configuración lo genera **svcutil**, por ejemplo:

```
<customBinding>

...

     <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>

...

</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

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.
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.

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 612). 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

Visión general de los servicios de Microsoft Windows Communication Foundation (WCF) que explican cómo crear y configurar servicios WCF.

El canal personalizado de IBM WebSphere 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 WebSphere 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 WCF de Microsoft, que contiene información adicional y los detalles más recientes, se puede encontrar en el sitio web de MSDN de Microsoft 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 617

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 WebSphere MQ 7.
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 619

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 WebSphere 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

Visión general de la generación y creación de aplicaciones cliente Microsoft Windows Communication Foundation (WCF).

Se puede crear una aplicación cliente para un servicio WCF; normalmente, las aplicaciones cliente se generan utilizando la herramienta Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) para crear los archivos de configuración y proxy necesarios que la aplicación puede utilizar directamente.

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 Microsoft svcutil.exe para generar un cliente para un servicio que está configurado para utilizar el canal personalizado WebSphere 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 WebSphere 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 el canal personalizado WebSphere MQ , pero también expone sus metadatos en tiempo de ejecución a través de un puerto HTTP independiente:

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` desde la raíz de la instalación al archivo de configuración svcutil activo, normalmente `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` para que svcutil reconozca el canal personalizado de WebSphere 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 626

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. Se deben realizar las modificaciones siguientes en el WSDL para especificar que se va a utilizar el canal personalizado WebSphere 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
```

   Donde *MQ_INSTALLATION_PATH* es el directorio de instalación de WebSphere MQ.

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 617

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` está en el directorio *`WMQInstallDir`*`\bin` donde *WMQInstallDir* es el directorio en el que está instalado WebSphere MQ 7.
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 619

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` 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 el enlace y la dirección de punto final configurados en el paso "1" en la página 627 y pasados:

   ```
    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) proporcionan algunos ejemplos sencillos de cómo se puede utilizar el canal personalizado de WebSphere MQ .

Para crear los proyectos de ejemplo, es necesario el SDK de Microsoft .NET 3.5 o Microsoft Visual Studio 2008.

Ejemplo sencillo de WCF servidor y cliente simple unidireccional

Este ejemplo muestra el canal personalizado de WebSphere MQ que se utiliza para iniciar un servicio de Windows Communication Foundation (WCF) desde un cliente WCF utilizando 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 625

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 WebSphere MQ. Para obtener más información sobre el formato del URI de punto final JMS, consulte *WebSphere MQ Transport for SOAP* en la documentación del producto WebSphere MQ . Si necesita modificar la solución de ejemplo y el origen, necesita un IDE, 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 WebSphere 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 WebSphere 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 muestra el canal personalizado de WebSphere MQ que se utiliza para iniciar un servicio de 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 625

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 debe 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 WebSphere MQ. Para obtener más información sobre el formato del URI de punto final JMS, consulte *WebSphere MQ Transport for SOAP* en la documentación del producto WebSphere MQ . Si necesita modificar la solución de ejemplo y el origen, necesita un IDE, 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 esperando 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 WebSphere MQ.
5. Ejecute el cliente una vez: 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 para WebSphere 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 a un servicio .NET alojado por WebSphere MQ

Se proporcionan aplicaciones cliente de ejemplo y aplicaciones proxy de servicio de ejemplo para .NET y 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 de alojamiento del servicio .NET SOAP sobre JMS esté correctamente instalado y configurado en WebSphere MQ y sea accesible desde un gestor de colas local. Para obtener información sobre la instalación y configuración del entorno, consulte: "Instalación del transporte web de WebSphere MQ para SOAP" en la página 976

Cuando el entorno de alojamiento del servicio .NET SOAP sobre JMS está correctamente instalado y configurado en WebSphere MQ y es accesible desde un gestor de colas local, deben realizarse pasos de configuración adicionales.

1. Establezca la variable de entorno WMQSOAP_HOME en el directorio de instalación de WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere MQ`
2. Asegúrese de que el compilador Java `javac` esté disponible y en la variable PATH.
3. Copie el archivo `axis.jar` del directorio `prereqs/axis` del CD de instalación de WebSphere en el directorio de producción WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere MQ\java\lib\soap`
4. Añada a la variable PATH: *MQ_INSTALLATION_PATH*`\Java\lib` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere MQ`
5. Asegúrese de que la ubicación de .NET se haya especificado correctamente en *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere 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 al directorio de trabajo 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 WebSphere MQ .
   - Para AXIS:*MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere 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 SOAP sobre JMS generado.
2. Suprima el gestor de colas.

Acerca de esta tarea

Este ejemplo muestra una conexión de un cliente WCF con el servicio de ejemplo .NET SOAP sobre JMS proporcionado en WebSphere MQ utilizando una forma 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 626

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 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 WebSphere MQ. Para obtener más información sobre el formato del URI de punto final JMS, consulte *WebSphere MQ Transport for SOAP* en la documentación del producto WebSphere 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 WebSphere 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.

Cliente WCF a un servicio Axis Java alojado en el ejemplo WebSphere MQ

Se proporcionan aplicaciones cliente de ejemplo y aplicaciones proxy de servicio de ejemplo para Java y .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 de alojamiento del servicio .NET SOAP sobre JMS esté correctamente instalado y configurado en WebSphere MQ y sea accesible desde un gestor de colas local. Para obtener información sobre la instalación y configuración del entorno, consulte: "Instalación del transporte web de WebSphere MQ para SOAP" en la página 976

Cuando el entorno de alojamiento del servicio .NET SOAP sobre JMS está correctamente instalado y configurado en WebSphere MQ y es accesible desde un gestor de colas local, deben realizarse pasos de configuración adicionales.

1. Establezca la variable de entorno WMQSOAP_HOME en el directorio de instalación de WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere MQ`
2. Asegúrese de que el compilador Java `javac` esté disponible y en la variable PATH.
3. Copie el archivo `axis.jar` del directorio `prereqs/axis` del CD de instalación de WebSphere en el directorio de instalación de WebSphere MQ .
4. Añada a la variable PATH: *MQ_INSTALLATION_PATH*\Java\lib donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere MQ`
5. Asegúrese de que la ubicación de .NET se haya especificado correctamente en *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ , por ejemplo: `C:\Program Files\IBM\WebSphere 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 al directorio de trabajo 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 WebSphere MQ .
- Para AXIS:*MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold donde *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere 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 SOAP sobre JMS generado.
2. Suprima el gestor de colas.

Acerca de esta tarea

El ejemplo muestra una conexión de un cliente WCF con el servicio de ejemplo Axis Java SOAP sobre JMS proporcionado en WebSphere MQ utilizando una forma 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 626

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 WebSphere 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 WebSphere 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.

Referencia relacionada

"Manejo de diferentes nombres de elementos de respuesta SOAP" en la página 641

WCF espera que el nombre de un valor devuelto tenga un formato específico de forma predeterminada, pero es posible que un servicio no devuelva un elemento con su nombre en el formato previsto.

Cliente WCF a servicio Java alojado por WebSphere Application Server de ejemplo

Se proporcionan aplicaciones cliente de ejemplo y aplicaciones proxy de servicio de ejemplo para WebSphere Application Server (WAS) 6. También se proporciona un servicio de solicitud y respuesta.

Antes de empezar

Este ejemplo requiere que se utilice la siguiente configuración de WebSphere MQ :

Tabla 77. WebSphere MQ configuración necesaria	
Objeto	**Nombre necesario**
Gestor de colas	QM1
Cola local	HelloWorld
Cola local	HelloWorldReply

Este ejemplo también requiere que un entorno de alojamiento de WebSphere Application Server V6 esté instalado y configurado correctamente. WebSphere Application Server V6 utiliza una conexión en modalidad de enlaces para conectarse a WebSphere MQ de forma predeterminada. Por lo tanto, WebSphere Application Server V6 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 denominado `HelloWorld`
      - Establezca el nombre JNDI en `jms/HelloWorld`
      - Establezca el nombre de cola en `HelloWorld`
   b. Una fábrica de conexiones de cola JMS denominada `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 denominada `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 denominado `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 la forma siguiente:
   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 WebSphere 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 al directorio de trabajo de Soap sobre JMS.
2. Especifique 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 WebSphere MQ.

Acerca de esta tarea

El ejemplo muestra una conexión de un cliente WCF con el servicio de ejemplo SOAP sobre JMS de WebSphere Application Server proporcionado en los ejemplos WCF incluidos en WebSphere MQ V7, utilizando una forma de canal de solicitud-respuesta. Los mensajes fluyen entre WCF y las colas de WebSphere Application Server utilizando WebSphere 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 625

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 debe cambiar el valor correspondiente en la aplicación cliente en el archivo *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\default\client\app.config` y en la aplicación de servicio en *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` donde *MQ_INSTALLATION_PATH* es el directorio de instalación de WebSphere MQ. Para obtener más información sobre el formato del URI de punto final JMS, consulte Sintaxis de URI y parámetros para el despliegue de servicio web .

El servicio y el cliente se basan en el servicio y el cliente descritos en el artículo de IBM Developer *Creación de un servicio web JMS utilizando SOAP sobre JMS y WebSphere Studio*. Si desea obtener más información sobre el desarrollo de servicios web SOAP sobre JMS compatibles con el canal personalizado WCF de WebSphere MQ , puede encontrar el artículo relevante en: https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Procedimiento

Ejecute 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 para WebSphere 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.

Determinación de problemas en el canal personalizado WCF para WebSphere MQ

Puede utilizar el rastreo de WebSphere MQ para recopilar información detallada sobre lo que están haciendo las distintas partes del código de WebSphere MQ . Cuando se utiliza Windows Communication Foundation (WCF), se genera una salida de rastreo independiente para el rastreo de canal personalizado WCF integrado con el rastreo de infraestructura WCF de Microsoft .

La habilitación completa del rastreo del canal personalizado WCF genera dos archivos de salida:

1. El rastreo de canal personalizado WCF integrado con el rastreo de infraestructura WCF de Microsoft .
2. El rastreo de canal personalizado WCF integrado con XMS .NET.

Al tener dos salidas de rastreo, se puede realizar un seguimiento de los problemas en cada interfaz utilizando las herramientas adecuadas, por ejemplo:

- Determinación de problemas WCF utilizando las herramientas de Microsoft adecuadas.
- WebSphere MQ Problemas del cliente MQI utilizando el formato de rastreo XMS .

Para simplificar la habilitación del rastreo, la pila de rastreo de .NET 3 TraceSource y XMS .NET se controlan utilizando una única interfaz tal como se describe en: "Configuración de rastreo WCF y nombres de archivo de rastreo" en la página 635.

Jerarquía de excepción del canal personalizado de WCF

Los tipos de excepciones que genera el canal personalizado son coherentes con WCF y normalmente son una excepción TimeoutException o CommunicationException (o una subclase de CommunicationException).

Se proporcionan más detalles sobre la condición de error, cuando proceda, utilizando excepciones enlazadas o internas. Las excepciones siguientes son ejemplos típicos, y cada capa de la arquitectura del canal aporta una excepción enlazada adicional, por ejemplo; CommunicationsException tiene una XMSException enlazada, que tiene una MQException enlazada:

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

La información de clave se captura y se proporciona en la recopilación de datos de la CommunicationException más alta de la jerarquía. Esta captura y suministro de datos evita la necesidad de que las aplicaciones se enlacen a cada capa en la arquitectura del canal para interrogar las excepciones enlazadas, y cualquier información adicional que puedan contener. Se definen los siguientes nombres de clave:

- IBM.XMS.WCF.ErrorCode: El código de mensaje de error de la excepción de canal personalizado actual.
- IBM.XMS.ErrorCode: El mensaje de error de la primera excepción XMS en la pila.
- IBM.WMQ.ReasonCode: El código de razón subyacente de WebSphere MQ .
- IBM.WMQ.CompletionCode: El código de terminación subyacente de WebSphere MQ .

Configuración de rastreo WCF y nombres de archivo de rastreo

Cuando el rastreo está totalmente habilitado, genera dos archivos de salida: uno para diagnosticar problemas de WCF y un archivo detallado para el material de diagnóstico de rastreo interno. Para simplificar la habilitación del rastreo, las pilas de rastreo .NET 3 TraceSource y XMS .NET utilizan una única interfaz.

Hay dos métodos de rastreo diferentes disponibles para el canal personalizado WCF, los dos métodos de rastreo se activan de forma independiente o juntos. Cada método produce su propio archivo de rastreo, de modo que cuando ambos métodos de rastreo están activados, se generan dos archivos de salida de rastreo.

Para mantener la configuración y la habilitación tan simples como sea posible, se utiliza la misma interfaz para controlar los dos métodos de rastreo. El archivo `app.config` se debe editar para incluir la configuración de rastreo relevante, tal como se describe en la siguiente sección. A continuación, los usuarios pueden añadir sus propias secciones equivalentes para combinar la salida con el rastreo de su propia aplicación.

El rastreo de canal personalizado WCF no está habilitado de forma predeterminada. En primer lugar, debe crear un escucha de rastreo y, a continuación, establecer el nivel de rastreo necesario para el origen de rastreo seleccionado en el archivo `app.config`.

Configuración del canal personalizado WCF con el rastreo de infraestructura WCF

Añada la siguiente sección de código a la sección `<system.diagnostics><sources>` en el archivo `app.config` :

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

El fragmento de código anterior realiza el rastreo de canal utilizando TraceSourcede .NET 3. Todas las invocaciones de los archivos de configuración asociados con los archivos ejecutables están controladas por este fragmento de código.

Configuración del canal personalizado WCF con rastreo XMS .NET

La configuración del rastreo de .NET de XMS requiere que añada una sección de código a la sección `<system.diagnostics><sources>` en el archivo `app.config` . Sin embargo, el fragmento de código se añade al elemento `<source>` extensible que se muestra en la sección Configuración del canal personalizado WCF con rastreo de infraestructura WCF . Por lo tanto, aunque el código de rastreo de la infraestructura WCF debe estar presente para que el rastreo de XMS .NET funcione, el rastreo de la infraestructura WCF puede inhabilitarse si no es necesario, tal como se describe en la sección Habilitación del rastreo WCF .

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

Variables de configuración del rastreo WCF

Tabla 78. Variables de configuración del rastreo WCF

Variable	**Descripción**
nombre	Especifique el nombre como: `IBM.XMS.WCF`
switchValue	switchValue controla el nivel de rastreo. Cuando switchValue se establece en `Off`, la infraestructura WCF TraceSource no se genera. Si se establece en cualquier otro valor, por ejemplo, `Verbose`, genera TraceSource. Para obtener información detallada sobre el nivel de rastreo de Microsoft, consulte la documentación de WCF o vaya a la página web Microsoft WCF Tracing: https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

Tabla 78. Variables de configuración del rastreo WCF (continuación)	
Variable	**Descripción**

xmsTraceSpecification =*ComponentName=type=state*	*Nombre_componente* es el nombre de la clase que desea rastrear. Puede utilizar un carácter comodín * en este nombre. Por ejemplo:
	`*=all=enabled`
	especifica que desea rastrear todas las clases, y
	`IBM.XMS.impl.*=all=enabled`
	especifica que solo necesita el rastreo de API.
	tipo puede ser cualquiera de los tipos de rastreo siguientes:
	• Todo
	• depurar
	• suceso
	• EntryExit
	estado puede ser habilitado o inhabilitado.

xmsTraceFilePath="*nombre_archivo*"	Si no especifica xmsTraceFilePath, o si xmsTraceFilePath está presente pero contiene una serie vacía, el archivo de rastreo se coloca en el directorio actual. Para almacenar el archivo de rastreo en un directorio con nombre, especifique el nombre del directorio en el archivo xmsTraceFilePath, por ejemplo: `xmsTraceFilePath="c:\somepath"`
xmsTraceFileSize="*tamaño*"	El tamaño máximo permitido del archivo de rastreo. Cuando un archivo alcanza este tamaño, se archiva y se renombra. El valor máximo predeterminado es 20 KB, que se especifica como: `xmsTraceFileSize="20000000".`
xmsTraceFileNumber="*número*"	El número de archivos de rastreo que se van a conservar. El valor predeterminado es 4 (un archivo activo y tres archivos de archivado). El número mínimo permitido es dos.

Tabla 78. Variables de configuración del rastreo WCF (continuación)	
Variable	**Descripción**

xmsTraceFormat="*formato*"	Hay dos niveles de xmsTraceFormat: `basic` y `advanced`. El formato de rastreo predeterminado es básico si no especifica xmsTraceFormat, o si xmsTraceFormat está presente pero contiene una serie vacía. Los archivos de rastreo se generan con este formato si especifica:
	`xmsTraceFormat="basic"`
	Si necesita un rastreo que sea compatible con las herramientas del analizador de rastreo, debe especificar:
	`traceFormat="advanced"`

Habilitación del rastreo WCF

Hay cuatro combinaciones para habilitar e inhabilitar los dos métodos de rastreo diferentes. Las cuatro combinaciones requieren la edición de los valores de las secciones del código descrito en las secciones precedentes.

También hay una variable de entorno que se puede establecer; para obtener más información, consulte “Habilitación del rastreo WCF con la variable de entorno WCF_TRACE_ON” en la página 639.

Esta tabla y los valores que se muestran dependen de los fragmentos de código demostrados previamente que ya se han añadido al archivo `app.config`.

Tabla 79. Combinaciones de habilitación de rastreo WCF.

Tipo de rastreo	Valor cambiado	Ejemplo
Rastreo XMS habilitado. WCF TraceSource habilitado.	`switchValue` no se ha establecido en `Off`	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Rastreo XMS habilitado. WCF TraceSource inhabilitado.	`switchValue` se ha establecido en `Off` y se ha proporcionado `xmsTraceSpecification`	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Tabla 79. Combinaciones de habilitación de rastreo WCF. (continuación)

Tipo de rastreo	Valor cambiado	Ejemplo
Rastreo XMS inhabilitado. WCF TraceSource habilitado.	Hay dos maneras de lograr este resultado: • La variable `switchValue` no se ha establecido en `Desactivado` y no se ha añadido un`xmsTraceSpecification` • La variable `switchValue` no se ha establecido en `Off` y `xmsTraceSpecification` se ha establecido en `disabled`	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Rastreo XMS inhabilitado. WCF TraceSource inhabilitado.	Hay tres maneras de lograr este resultado: • No hay ningún elemento `<source>` en el archivo `app.config` • La variable `switchValue` se establece en `Desactivado` y no se ha añadido un`xmsTraceSpecification` • La variable `switchValue` se ha establecido en `Off` y `xmsTraceSpecification` se ha establecido en `disabled`	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Habilitación del rastreo WCF con la variable de entorno WCF_TRACE_ON

Además de los métodos anteriores descritos para habilitar el rastreo de WCF, el rastreo de XMS .NET también se puede habilitar utilizando la variable de entorno WCF_TRACE_ON.

Establecer la variable de entorno WCF_TRACE_ON en cualquier valor no nulo es el equivalente a establecer `xmstraceSpecification` en `*=all=enabled`, por ejemplo: "`set WCF_TRACE_ON=true`"

No obstante, si `xmstraceSpecification` se establece explícitamente en el archivo `app.config`, la variable de entorno WCF_TRACE_ON se altera temporalmente.

Archivos de salida de rastreo WCF y nombres de archivo

Los archivos de rastreo XMS se denominan tradicionalmente utilizando un formato de nombre base e ID de proceso: `xms_trace_`*`pid`*`.log`, donde *pid* es el ID de proceso.

Puesto que los archivos de rastreo XMS todavía se pueden producir en paralelo con los archivos de rastreo de canal personalizado WCF, el rastreo de canal personalizado WCF integrado con los archivos de salida de rastreo XMS .NET tienen el formato siguiente para evitar confusiones: `wcfxms_trace_`*`pid`*`.log`, donde *pid* es el ID de proceso.

De forma predeterminada, el archivo de salida de rastreo se crea en el directorio de trabajo actual, pero este destino se puede redefinir si es necesario.

WCF XMS First Failure Support Technology (FFST)

Puede recopilar información detallada sobre lo que están haciendo varias partes del código de WebSphere MQ utilizando el rastreo de WebSphere MQ . XMS FFST tiene sus propios archivos de configuración y salida para el canal personalizado WCF.

Los archivos de rastreo XMS FFST se denominan tradicionalmente utilizando el nombre base y el formato de ID de proceso de: `xmsffdc`*`pid_date`*`.txt`, donde *pid* es el ID de proceso y *fecha* es la hora y la fecha.

Puesto que los archivos de rastreo XMS FFST todavía se pueden producir en paralelo con los archivos XMS FFST del canal personalizado WCF, los archivos de salida XMS FFST del canal personalizado WCF tienen el formato siguiente para evitar confusiones: `wcfffdc`*`pid_date`*`.txt`, donde *pid* es el ID de proceso y *fecha* es la fecha y hora.

De forma predeterminada, este archivo de salida de rastreo se crea en el directorio de trabajo actual, pero este destino se puede redefinir si fuera necesario.

El canal personalizado WCF con la cabecera de rastreo XMS .NET es similar al ejemplo siguiente:

```
************* Start Display XMS WCF Environment *************
  Product Name  :- value
  WCF Version   :- value
  Level    :- value
*************  End Display XMS WCF Environment  *************
```

Los archivos de rastreo FFST se formatean de la forma estándar, sin ningún formato que sea específico del canal personalizado.

Información de versión de WCF

La información de versión de WCF ayuda a determinar problemas y se incluye en los comandos de ensamblaje del canal personalizado.

El canal personalizado de WebSphere MQ para metadatos de versión WCF se puede recuperar de una de estas tres maneras:

- Utilizando el programa de utilidad WebSphere MQ dspmqver. Para obtener más información sobre cómo utilizar dspmqver, consulte: dspmqver
- Utilizando el diálogo de propiedades del Explorador de Windows : en el Explorador de Windows , pulse con el botón derecho del ratón en **IBM.XMS.WCF.dll** > **Propiedades** > **Versión**.
- Desde la información de cabecera de cualquiera de los canales FFST o archivos de rastreo. Para obtener más información sobre la información de cabecera FFST , consulte: “WCF XMS First Failure Support Technology (FFST)” en la página 639

Sugerencias y consejos de WCF

Los consejos y sugerencias siguientes no siguen ningún orden en particular y podrían añadirse más cuando se publiquen nuevas versiones de la documentación. Son cuestiones que, si son relevantes para el trabajo que está realizando, pueden ahorrarle tiempo.

Externalización de excepciones desde el host de servicio WCF

En el caso de los servicios alojados utilizando el host de servicio WCF, las excepciones no manejadas lanzadas por el servicio, los internos de WCF o la pila de canales no se externalizan de forma predeterminada. Para ser notificado de estas excepciones, hay que registrar un manejador de errores.

El código siguiente proporciona un ejemplo de definición del comportamiento de un servicio manejador de errores que se puede aplicar como un atributo de un servicio:

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, Collection<ServiceEndpoint> endpoints,
           BindingParameterCollection bindingParameters)
```

```
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        //
        // IErrorHandler Interface
        //
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
        }
    }
```

Manejo de diferentes nombres de elementos de respuesta SOAP

WCF espera que el nombre de un valor devuelto tenga un formato específico de forma predeterminada, pero es posible que un servicio no devuelva un elemento con su nombre en el formato previsto.

WCF tiene el convenio de esperar que el valor devuelto se denomine con el formato siguiente: *methodName*`Result` donde *methodName* es el nombre de la operación de servicio. Por ejemplo, para un servicio denominado `getQuote`, WCF espera que se llame a la respuesta: `getQuoteResult`.

No obstante, el servicio puede devolver un elemento con un nombre que no se ajusta a este formato.

Cuando se ejecuta la herramienta scvutil para generar un cliente proxy, si WSDL especifica un nombre diferente, la interfaz de proxy añade los parámetros para indicar a WCF el nombre que ha de buscar. Por ejemplo:

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style =
System.ServiceModel.OperationFormatStyle.Rpc,
                                                  Use =
System.ServiceModel.OperationFormatUse.Encoded)]
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

Si crea su propia interfaz, por ejemplo, añadiendo un método de solicitud/respuesta a una interfaz de proxy existente, debe asegurarse de que añade los mismos parámetros a la interfaz si el servicio devuelve un nombre diferente. Si no lo hace, el problema más común es que una llamada a un método de servicio devuelva siempre un valor nulo. Si se devuelve un objeto, el método devuelve un nulo, pero si se devuelve un valor numérico, tal como un entero, el método devuelve 0.

Utilización de C++

WebSphere MQ proporciona clases C++ equivalentes a los objetos WebSphere MQ y algunas clases adicionales equivalentes a los tipos de datos de matriz. Proporciona una serie de características que no están disponibles en MQI.

A partir de WebSphere MQ Versión 7.0, las mejoras en las interfaces de programación de WebSphere MQ no se aplicarán a las clases C++.

WebSphere MQ C++ proporciona las características siguientes:

- Inicialización automática de estructuras de datos WebSphere 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 cabecera de puente IMS .
- Recepción y transmisión de la cabecera del mensaje de referencia.
- Recepción de mensajes de desencadenantes.
- Recepción y transmisión de cabecera de puente CICS .
- Recepción y transmisión de la cabecera de trabajo.
- Definición de canal de cliente.

Los siguientes diagramas de clase Booch muestran que todas las clases son en general paralelas a las entidades WebSphere MQ de la MQI de procedimiento (por ejemplo, utilizando C) que tienen manejadores o estructuras de datos. Todas las clases heredan de la clase ImqError (consulte ImqError clase C++), lo que permite asociar una condición de error con cada objeto.

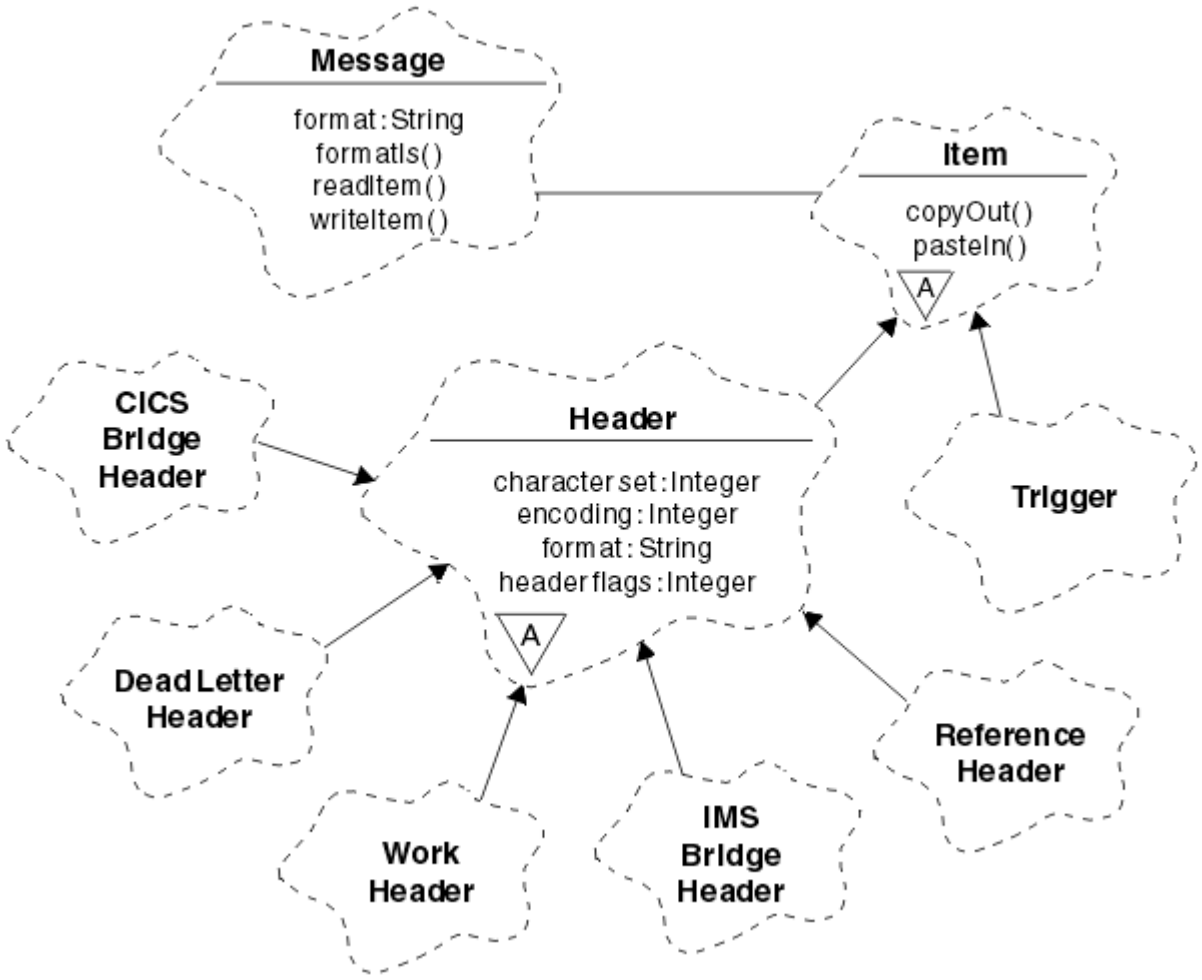


Figura 120. Clases C++ de WebSphere MQ (manejo de elementos)

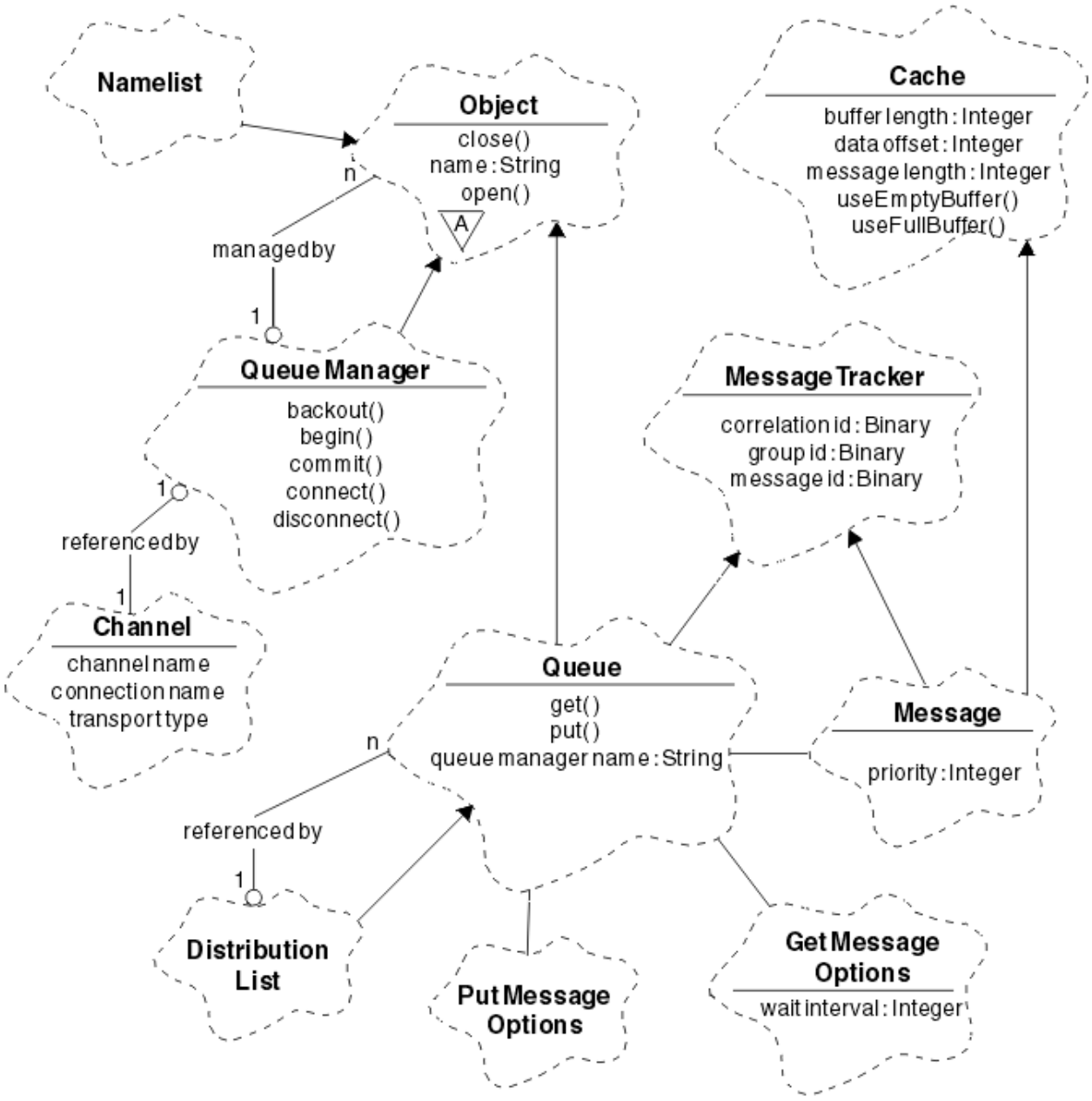


Figura 121. Clases C++ de WebSphere MQ (gestión de colas)

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 denota 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 121 en la página 643) y las clases de gestión de elementos (consulte Figura 120 en la página 642):

- La clase ImqBinary (consulte ImqBinary clase C++), que encapsula matrices de bytes como MQBYTE24.
- El tipo de datos ImqBoolean, que se define como **typedef unsigned char ImqBoolean**.

- La clase ImqString (consulte ImqString C++ class), 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 descriptores de contexto se encuentran bajo la jerarquía de clases ImqObject (consulte ImqObject clase C++) 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 ImqMessage clase C++) encapsula la estructura de datos MQMD y también actúa como punto de retención para datos de usuario y *elementos* (consulte “Lectura de mensajes en C++” en la página 653) 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 644
Se proporcionan cuatro programas de ejemplo que muestran cómo obtener y transferir mensajes.

“consideraciones relativas a C++” en la página 648
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 652
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.

“Decidir qué lenguaje de programación utilizar” en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

“Desarrollo de aplicaciones” en la página 7
IBM WebSphere MQ proporciona varias formas en las que puede desarrollar aplicaciones para enviar y recibir mensajes que necesita para dar soporte a los procesos de negocio. También puede desarrollar aplicaciones para gestionar los gestores de colas y recursos relacionados.

Referencia relacionada

“Creación de programas C++ de WebSphere MQ” en la página 659
Se lista el URL de los compiladores soportados, junto con los mandatos a utilizar para compilar, enlazar y ejecutar programas y ejemplos C++ en plataformas WebSphere MQ .

Referencia cruzada de C++ y MQI

Clases C++ de WebSphere 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 80 en la página 645.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Tabla 80. 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
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ah (Consulte la nota "2" en la página 645)
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn (Consulte la nota "3" en la página 645)

Notas:

1. Los programas compilados con el compilador ILE C++ para IBM i están en la biblioteca QMQM. Los archivos de inclusión están en /QIBM/ProdData/mqm/inc.
2. Los programas compilados con el compilador HP ANSI C++ están en el directorio *MQ_INSTALLATION_PATH*/samp/bin/ah. Para obtener más información, consulte "Compilación de programas C++ en HP-UX" en la página 660.
3. Los programas creados utilizando Microsoft Visual Studio se encuentran en *MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn. Para obtener más información sobre estos compiladores, consulte "Creación de programas C++ en Windows" en la página 665.

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 está utilizando una conexión de servidor con WebSphere 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 está utilizando una conexión de cliente con WebSphere MQ, siga uno de los procedimientos siguientes:

- Configure la variable de entorno MQSERVER (consulte MQSERVER para obtener más información) y ejecute **imqwrldc**, o bien
- Ejecute **imqwrldc** pasando como parámetros **queue-name**, **queue-manager-name**y **channel-definition**, donde un **channel-definition** típico podría ser SYSTEM.DEF.SVRCONN/TCP/*nombre_host*(1414)

Código de ejemplo

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // WebSphere MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );
```

```
      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

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 ImqError clase C++)
- ImqMessage (consulte ImqMessage ImqMessage)
- ImqObject (consulte ImqObject clase C++)

- ImqQueue (consulte ClaseImqQueue C++)
- ImqQueueManager (consulte ImqQueueManager)

Siga las instrucciones adecuadas para ejecutar los programas.

En todas las plataformas excepto z/OS

1. Ejecute **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 **imqsgets** *nombre-cola* para recuperar todas las líneas y visualizarlas en la consola.

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 está soportado 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 los mensajes de esas 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 la MQI, para ayudarle a escribir programas de aplicación WebSphere MQ en el lenguaje C++.

Estos archivos de cabecera se resumen en la tabla siguiente.

Tabla 81. 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**
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 con 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. Sin embargo, WebSphere MQ impone determinados límites que pueden hacer que la información se trunque. 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 las funciones C se pueden utilizar en **char ***, hay métodos especiales de la clase **ImqString** que son preferibles; **longitud de operador**() es el equivalente de **strlen** y **storage**() indica la memoria asignada para los datos de tipo carácter.

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

Un objeto de gestor ImqQueuese conecta automáticamente para cualquier método que dé como resultado una llamada a la MQI (consulte Referencia cruzada de C++ y MQI).

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 datos de carácter (**char ***) siempre toman una copia de los datos, pero algunos métodos pueden truncar la copia, porque WebSphere MQimpone determinados límites.

La clase ImqString (consulte ImqString clase C++) encapsula el carácter tradicional **char *** 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 ImqBinary clase C++) encapsula 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:

- Cabecera ImqCICSBridge(consulte ImqCICSBridgeclase C++ de cabecera)
- ImqGetMessageOptions (consulte ImqGetMessageOptions clase C++)
- Cabecera ImqIMSBridge(consulte ImqIMSBridgeclase C++ de cabecera)
- ImqMessageTracker (consulte ImqMessageTracker C++ class)
- ImqQueueManager (consulte ImqQueueManager)
- Cabecera ImqReference(consulte ImqReferenceclase C++ de cabecera)
- Cabecera ImqWork(consulte Clase C++ de cabeceraImqWork)

La clase ImqBinary también proporciona soporte para la comparación y la copia.

Funciones no soportadas en C++

Las clases y métodos C++ de WebSphere MQ son independientes de la plataforma WebSphere MQ . Por lo tanto, es posible que ofrezcan algunas funciones que no estén soportadas en determinadas plataformas.

Si intenta utilizar una función en una plataforma en la que no está soportada, la función es detectada por WebSphere MQ pero no por los enlaces de lenguaje C++. WebSphere MQ informa del 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 envía un mensaje, los datos de mensaje se preparan por primera vez en un almacenamiento intermedio gestionado por un objeto ImqCache (consulte ImqCache Clase C++). Un almacenamiento intermedio se asocia (por herencia) con cada objeto ImqMessage (consulte la clase C++ImqMessage): lo puede proporcionar la aplicación (utilizando el método **useEmptyBuffer** 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 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 reconocible de WebSphere MQ . 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 ImqItem clase C++) e implementa los métodos virtuales **copyOut** y **pasteIn** de dicha clase.

Los dos ejemplos siguientes muestran código de la clase ImqDeadLetterHeader (consulte ImqDeadLetterHeader C++ class). El primer ejemplo muestra el código de*escritura* de mensajes encapsulados personalizados.

```
// 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 personalizados.

```
// 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 WebSphere MQ-IMS pueden utilizar una cabecera especial. La cabecera de puente IMS tiene como prefijo los datos de mensaje 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 );
```

Grabación de un mensaje en el puente CICS en C++

Código de programa de ejemplo para escribir un mensaje en el puente CICS .

Los mensajes enviados a WebSphere MQ para z/OS utilizando el puente CICS requieren una cabecera especial. La cabecera de puente CICS tiene como prefijo los datos de mensaje 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 WebSphere MQ para z/OS, que están destinados a una cola gestionada por z/OS Workload Manager, 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 C++ de WebSphere MQ

Se lista el URL de los compiladores soportados, junto con los mandatos a utilizar para compilar, enlazar y ejecutar programas y ejemplos C++ en plataformas WebSphere MQ .

Los compiladores para cada plataforma y versión soportadas de WebSphere MQ se listan en la página de requisitos del sistema de WebSphere MQ en Requisitos del sistema para IBM WebSphere MQ.

El mandato que necesita para compilar y enlazar el programa C++ de WebSphere MQ depende de la instalación y de los requisitos. Los ejemplos siguientes muestran mandatos de compilación y enlace típicos para algunos de los compiladores que utilizan la instalación predeterminada de WebSphere MQ en varias plataformas.

Creación de programas C++ en AIX

Cree programas WebSphere MQ C++ en AIX utilizando el compilador XL C Enterprise Edition .

Cliente

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Aplicación sin hebras de 32 bits

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Aplicación con hebras de 32 bits

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Aplicación sin hebras de 64 bits

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Aplicación con hebras de 64 bits

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

Servidor

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Aplicación sin hebras de 32 bits

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Aplicación con hebras de 32 bits

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Aplicación sin hebras de 64 bits

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Aplicación con hebras de 64 bits

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

Compilación de programas C++ en HP-UX

Cree programas WebSphere MQ C++ en HP-UX utilizando los compiladores aC+ + o aCC .

En HP-UX Itanium, WebSphere MQ sólo da soporte al tiempo de ejecución estándar. Utilice el compilador de aCC.

- libimqi23bh.sl proporciona las clases C++ de WebSphere MQ para el tiempo de ejecución estándar.
- Para la compatibilidad con releases anteriores, se proporciona un enlace simbólico de libimqi23ah.sl a libimqi23bh.sl.

IA64 (IPF)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en que está instalado WebSphere MQ.

Cliente: IA64 (IPF)

Aplicación sin hebras de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

Aplicación con hebras de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

Aplicación sin hebras de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqic
```

Aplicación con hebras de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

Servidor: IA64 (IPF)

Aplicación sin hebras de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

Aplicación con hebras de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

Aplicación sin hebras de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

Aplicación con hebras de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

Compilación de programas C++ en Linux

Cree programas WebSphere MQ C++ en Linux utilizando el compilador GNU g + +.

System p

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Cliente: System p

Aplicación sin hebras de 32 bits

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Servidor: System p

Aplicación sin hebras de 32 bits

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

System z

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Cliente: System z

Aplicación sin hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64  -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Servidor: System z

Aplicación sin hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Sistema x (32 bits)

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Cliente: System x (32 bits)

Aplicación sin hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,
-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Aplicación con hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Servidor: System x (32 bits)

Aplicación sin hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATHlib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Aplicación sin hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Aplicación con hebras de 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Creación de programas C++ en Solaris

Cree programas C++ de WebSphere MQ en Solaris utilizando el compilador Sun ONE.

SPARC

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Cliente: SPARC

Aplicación de 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicación de 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Servidor: SPARC

Aplicación de 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplicación de 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Cliente: x86-64

Aplicación de 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicación de 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Servidor: x86-64

Aplicación de 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplicación de 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

Creación de programas C++ en Windows

Cree programas C++ de WebSphere MQ en Windows utilizando el compilador Microsoft Visual Studio C++.

Los archivos de biblioteca (.lib) y los archivos dll para utilizarlos con aplicaciones de 32 bits se instalan en *MQ_INSTALLATION_PATH*\Tools\Lib, los archivos para utilizarlos con aplicaciones de 64 bits se instalan en *MQ_INSTALLATION_PATH*\Tools\Lib64. *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ .

Cliente

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

Servidor

```
 cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Utilización de clases de WebSphere MQ para Java

WebSphere MQ classes for Java le permite utilizar WebSphere MQ en un entorno Java. Una aplicación Java puede utilizar clases WebSphere MQ para Java o clases WebSphere MQ para JMS para acceder a recursos WebSphere MQ .

Las clases de WebSphere MQ para Java permiten que una aplicación Java:

- Conéctese a WebSphere MQ como un cliente WebSphere MQ
- Conectar directamente con un gestor de colas de WebSphere MQ

WebSphere MQ classes for Java encapsulan la interfaz de cola de mensajes (MQI), la API nativa de WebSphere MQ .

Las clases WebSphere MQ para Java utilizan un modelo de objeto similar a las interfaces C++ y .NET para WebSphere MQ.

¿Por qué debo utilizar las clases WebSphere MQ para Java?

Si los puntos siguientes son significativos en la instalación, considere la posibilidad de utilizar las clases WebSphere MQ para Java:

- WebSphere MQ classes for Java encapsulan la interfaz de cola de mensajes (MQI), la API nativa de WebSphere MQ .
  - Si está familiarizado con el uso de MQI en lenguajes de procedimiento, puede transferir este conocimiento al entorno Java.
  - Puede utilizar toda la gama de características de WebSphere MQ, más allá de las disponibles a través de JMS.
- Las clases WebSphere MQ para Java utilizan un modelo de objeto similar a las interfaces C++ y .NET para WebSphere MQ. Si está familiarizado con estas interfaces, puede transferir este conocimiento al entorno Java.

Nota: La reconexión automática de cliente no está soportada por las clases WebSphere MQ para Java.

Iniciación a las clases WebSphere MQ para Java

Esta colección de temas proporciona una visión general de las clases de WebSphere MQ para Java y sus usos.

¿Qué son las clases de WebSphere MQ para Java?

Las clases WebSphere MQ para Java le permiten utilizar WebSphere MQ en un entorno Java.

Las clases de WebSphere MQ para Java permiten que una aplicación Java:

- Conéctese a WebSphere MQ como un cliente WebSphere MQ
- Conectar directamente con un gestor de colas de WebSphere MQ

WebSphere MQ classes for Java encapsulan la interfaz de cola de mensajes (MQI), la API nativa de WebSphere MQ .

Las clases WebSphere MQ para Java utilizan un modelo de objeto similar a las interfaces C++ y .NET para WebSphere MQ.

¿Por qué debo utilizar las clases WebSphere MQ para Java?

Una aplicación Java puede utilizar clases WebSphere MQ para Java o clases WebSphere MQ para JMS para acceder a recursos WebSphere MQ . Existen varias ventajas al utilizar clases de WebSphere MQ para Java.

Si los puntos siguientes son significativos en la instalación, considere la posibilidad de utilizar clases de Websphere MQ para Java:

- WebSphere MQ classes for Java encapsulan la interfaz de cola de mensajes (MQI), la API nativa de WebSphere MQ .
  - Si está familiarizado con el uso de MQI en lenguajes de procedimiento, puede transferir este conocimiento al entorno Java.
  - Puede utilizar toda la gama de características de WebSphere MQ, más allá de las disponibles a través de JMS.
- Las clases WebSphere MQ para Java utilizan un modelo de objeto similar a las interfaces C++ y .NET para WebSphere MQ. Si está familiarizado con estas interfaces, puede transferir este conocimiento al entorno Java.

Opciones de conexión para clases de WebSphere MQ para Java

WebSphere MQ classes for Java puede conectarse en modalidad de cliente o de enlaces.

Las opciones programables permiten que WebSphere MQ classes for Java se conecte a WebSphere MQ de una de las maneras siguientes:

- Como cliente MQI de WebSphere MQ utilizando protocolo de control de transmisiones/Internet Protocol (TCP/IP)
- En modalidad de enlaces, se conecta directamente a WebSphere MQ utilizando JNI (Java Native Interface)

Los clientes no se pueden ejecutar en z/OS, pero los clientes de otras plataformas se pueden conectar a un gestor de colas de WebSphere MQ for z/OS si está instalado Client Attach Facility.

En las secciones siguientes se describen más detalladamente las opciones de conexión en modalidad de cliente y en modalidad de enlaces.

Conexión de cliente

Para conectarse a un gestor de colas en modalidad de cliente, una aplicación WebSphere MQ classes for Java puede ejecutarse en el mismo sistema en el que se ejecuta el gestor de colas, o en un sistema diferente. En cada caso, WebSphere MQ classes for Java se conecta al gestor de colas a través de TCP/IP.

Una aplicación WebSphere MQ para Java puede conectarse a cualquier gestor de colas soportado utilizando la modalidad de cliente.

Para obtener más información sobre cómo escribir aplicaciones para utilizar conexiones en la modalidad de cliente, consulte “WebSphere Clases MQ para modalidades de conexión Java” en la página 681.

conexión de enlaces

Cuando se utiliza en modalidad de enlaces, WebSphere MQ classes for Java utiliza JNI (Java Native Interface) para llamar directamente a 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 modalidad de enlaces proporciona

un mejor rendimiento para las clases de WebSphere MQ para aplicaciones Java que la conexión en modalidad de cliente, evitando el coste de la comunicación TCP/IP.

Las aplicaciones que utilizan las clases de WebSphere MQ para que Java se conecte en modalidad de enlaces deben ejecutarse en el mismo sistema que el gestor de colas al que se conectan.

Java Runtime Environment, que se utiliza para ejecutar las clases de WebSphere MQ para la aplicación Java, debe estar configurado para cargar las clases de WebSphere MQ para bibliotecas Java; consulte Las clases de WebSphere MQ para bibliotecas Java para obtener más información.

Para obtener más información sobre cómo escribir aplicaciones para utilizar conexiones en la modalidad de enlaces, consulte “WebSphere Clases MQ para modalidades de conexión Java” en la página 681.

Requisitos previos para las clases de WebSphere MQ para Java

Para utilizar WebSphere MQ classes for Java, necesita otros productos de software.

Para obtener la información más reciente sobre los requisitos previos para WebSphere MQ classes for Java, consulte el archivo README de WebSphere MQ .

Para desarrollar WebSphere MQ clases para aplicaciones Java, necesita un Java Development Kit (JDK). Los detalles de los JDK soportados con el sistema operativo se pueden encontrar en la página de requisitos del sistema de WebSphere MQ en Requisitos del sistema para IBM WebSphere MQ.

Para ejecutar las clases de WebSphere MQ para aplicaciones Java, necesita los siguientes componentes de software:

- Un gestor de colas de WebSphere MQ , para aplicaciones que se conectan a un gestor de colas
- Un Java Runtime Environment (JRE) por cada sistema en el que se ejecutan aplicaciones. Se proporciona un JRE adecuado con WebSphere MQ.

Si necesita conexiones SSL para utilizar módulos criptográficos que han sido certificados por FIPS 140-2, necesita el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS). Cada IBM JDK y JRE de la versión 1.4.2 o posterior contiene IBMJSSEFIPS.

Puede utilizar las direcciones de Internet Protocol Versión 6 (IPv6) en las clases WebSphere MQ para aplicaciones Java si IPv6 está soportado por la máquina virtual Java (JVM) y la implementación TCP/IP en el sistema operativo.

Instalación y configuración de clases de WebSphere MQ para Java

En esta sección se describen los directorios y archivos que se crean al instalar WebSphere MQ classes for Java, y se explica cómo configurar WebSphere MQ classes for Java después de la instalación.

Qué se instala para WebSphere MQ classes for Java

La versión más reciente de WebSphere MQ classes for Java se instala con WebSphere 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 WebSphere MQ , consulte:

Instalación de un servidor WebSphere MQ
Instalación de un cliente de IBM WebSphere MQ

Las clases WebSphere MQ para Java están contenidas en los archivos JAR (Java Archive), com.ibm.mq.jary com.ibm.mq.jmqi.jar.

El soporte para cabeceras de mensajes estándar, como por ejemplo PCF (Programmable Command Format), 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.

Instalación y actualización de las clases WebSphere MQ para archivos JAR Java

La única forma soportada de obtener las clases WebSphere MQ para archivos JAR Java en un sistema es instalar el producto WebSphere MQ o el cliente WebSphere MQ MQI SupportPac, o utilizar una herramienta de gestión de software como Apache Maven, para obtener más información, consulte “IBM WebSphere MQ classes for Java y herramientas de gestión de software” en la página 677.

No mueva ni copie las clases de WebSphere MQ para archivos JAR Java de otras máquinas, a menos que esté utilizando una herramienta de gestión de software.

- Los fixpacks no se pueden aplicar a una "instalación" donde los archivos JAR se han copiado desde otra máquina, y hace que sea mucho más difícil asegurarse de que todos los archivos JAR se mantienen en paso uno con el otro, y están en niveles compatibles.
- La copia de las clases WebSphere MQ para archivos JAR JMS entre máquinas también puede dar como resultado varias copias de los archivos que residen en la misma máquina, lo que puede provocar problemas al dar servicio al código y depurar problemas.

No incluya las clases de WebSphere MQ para archivos JAR Java dentro de los archivadores de aplicación.

- Las actualizaciones de las clases de WebSphere MQ para Java no se pueden aplicar utilizando un fixpack de WebSphere MQ .
- El soporte de IBM no puede determinar fácilmente la versión de las clases de WebSphere MQ para Java que utiliza la aplicación.
- Pueden surgir problemas si varias aplicaciones que se ejecutan en el mismo Java Runtime Environment incluyen distintas versiones de las clases WebSphere MQ para Java, ya que varias versiones de las clases WebSphere MQ para Java se cargan en el Java Runtime Environment al mismo tiempo.
- Si una aplicación utiliza el transporte BINDINGS para conectarse a un gestor de colas, cualquier actualización importante del gestor de colas también requiere que la aplicación se actualice para incluir el nivel correspondiente de las clases WebSphere MQ para Java.

  Por ejemplo, si un gestor de colas se actualiza al nivel WebSphere MQ Versión 7.1 , las aplicaciones que se conectan al gestor de colas utilizando el transporte BINDINGS también deben actualizarse para incluir las clases WebSphere MQ Versión 7.1 para Java.

La biblioteca Java siguiente se distribuye con las clases WebSphere MQ para Java:

- connector.jar (Versión 1.0)

La aplicación de ejemplo denominada Postales se encuentra en el archivo JAR com.ibm.mq.postcard.jar.

La herramienta Javadoc se ha utilizado para generar las páginas HTML que contienen las especificaciones de las clases WebSphere MQ para Java y las clases WebSphere MQ para las API JMS. Las páginas HTML se encuentran en el subdirectorio doc del directorio de instalación de WebSphere MQ classes for JMS. En los sistemas UNIX, Linuxy Windows , el subdirectorio doc contiene las páginas HTML individuales.

Cuando finalice la instalación, los archivos y ejemplos se instalarán en las ubicaciones que se muestran en la “Directorios de instalación para WebSphere MQ classes for Java” en la página 669.

Después de la instalación, en cualquier plataforma que no sea Windows, debe actualizar las variables de entorno tal como se describe en “Variables de entorno relevantes para WebSphere MQ classes for Java” en la página 670.

Directorios de instalación para WebSphere MQ classes for Java

WebSphere Las clases MQ para archivos Java se instalan en distintas ubicaciones según la plataforma.

La Tabla 82 en la página 670 muestra dónde se instalan las clases de WebSphere MQ para archivos Java.

Tabla 82. Directorios de instalación de WebSphere MQ para Java	
Plataforma	**Directorio**
AIX	*MQ_INSTALLATION_PATH*/java/lib
HP-UX, Linuxy Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.	

Algunas aplicaciones de ejemplo, como los programas de verificación de la instalación (IVP), se proporcionan con WebSphere MQ. Tabla 83 en la página 670 muestra dónde se instalan las aplicaciones de ejemplo. Las clases de WebSphere MQ para ejemplos de Java se encuentran en un subdirectorio denominado `wmqjava`. Los ejemplos de PCF se encuentran en un subdirectorio llamado `pcf`.

Tabla 83. Directorios de ejemplos	
Plataforma	**Directorio**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
HP-UX, Linuxy Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.	

Variables de entorno relevantes para WebSphere MQ classes for Java

Si desea ejecutar clases WebSphere MQ para aplicaciones Java, sus vías de acceso de clases deben incluir las clases WebSphere MQ para Java y directorios de ejemplos.

Para que se ejecuten las clases de WebSphere MQ para aplicaciones Java, su vía de acceso de clases debe incluir las clases de WebSphere MQ adecuadas para el directorio Java. 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.

La Tabla 84 en la página 670 muestra el valor de CLASSPATH adecuado para utilizar en cada plataforma para ejecutar clases de WebSphere MQ para aplicaciones Java, incluidas las aplicaciones de ejemplo.

Tabla 84. Valor CLASSPATH para ejecutar las clases WebSphere MQ para aplicaciones Java, incluidas las clases WebSphere MQ para aplicaciones de ejemplo Java	
Plataforma	**Valor de CLASSPATH**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
HP-UX, Linuxy Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.	

Si compila utilizando la opción -Xlint, puede recibir un mensaje para avisarle de que com.ibm.mq.ese.jar no está presente. Puede pasar por alto este aviso. Este archivo sólo está presente si ha instalado IBM WebSphere MQ Advanced Message Security.

Los scripts proporcionados con WebSphere MQ classes for Java 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 se instalan las clases de WebSphere MQ para Java, tal como se muestra en Directorios de instalación deWebSphere MQ para Java.

MQ_JAVA_LIB_PATH
Esta variable de entorno especifica el directorio donde se almacenan las clases de WebSphere MQ para bibliotecas Java, tal como se muestra en Ubicación de las clases de WebSphere MQ para bibliotecas Java para cada plataforma. Algunos scripts proporcionados con WebSphere MQ classes for Java, como IVTRun, utilizan esta variable de entorno.

En Windows, todas las variables de entorno se establecen automáticamente durante la instalación. En cualquier otra plataforma, debe establecerlas usted mismo.En un sistema UNIX , puede utilizar el script **setjmsenv** (si está utilizando una JVM de 32 bits) o **setjmsenv64** (si está utilizando una JVM de 64 bits) para establecer las variables de entorno. En AIX, HP-UX, Linuxy Solaris, estos scripts se encuentran en el directorio *MQ_INSTALLATION_PATH*/java/bin .

Las clases IBM WebSphere MQ para bibliotecas Java

La ubicación de las clases IBM WebSphere MQ para bibliotecas Java varía según la plataforma. Especifique esta ubicación cuando inicie una aplicación.

Para especificar la ubicación de las bibliotecas JNI (Java Native Interface), inicie la aplicación utilizando un mandato **java** con el formato siguiente:

```
java -Djava.library.path=library_path application_name
```

donde *vía_acceso_biblioteca* es la vía de acceso a las clases de WebSphere MQ para bibliotecas Java, que incluyen las bibliotecas JNI. Tabla 85 en la página 671 muestra la ubicación de las clases de WebSphere MQ para bibliotecas Java para cada plataforma.

Tabla 85. La ubicación de las clases de WebSphere MQ para bibliotecas Java para cada plataforma

Plataforma	**Directorio que contiene las clases de WebSphere MQ para bibliotecas Java**
AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
HP-UX Linux (POWER, x86-64 y zSeries s390x plataformas) Solaris (plataformasx86-64 y SPARC)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Linux (plataformax86)	*MQ_INSTALLATION_PATH*/java/lib

Tabla 85. La ubicación de las clases de WebSphere MQ para bibliotecas Java para cada plataforma (continuación)	
Plataforma	**Directorio que contiene las clases de WebSphere MQ para bibliotecas Java**
Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliotecas de 64 bits)
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.	

Nota:

1. En AIX, HP-UX, Linux (Power platform) o Solaris, utilice las bibliotecas de 32 bits o las bibliotecas de 64 bits. Utilice las bibliotecas de 64 bits sólo si está ejecutando la aplicación en una máquina virtual Java (JVM) de 64 bits en una plataforma de 64 bits. De lo contrario, utilice las bibliotecas de 32 bits.
2. En Windows, puede utilizar la variable de entorno PATH para especificar la ubicación de las bibliotecas de WebSphere MQ para Java en lugar de especificar su ubicación en el mandato **java** .
3. Para utilizar WebSphere MQ classes for Java en modalidad de enlaces en IBM i, asegúrese de que la biblioteca QMQMJAVA esté en la lista de bibliotecas.

Tareas relacionadas

Utilización de clases de WebSphere MQ para Java

Soporte para OSGi en IBM WebSphere MQ classes for Java

OSGi proporciona una infraestructura que da soporte al despliegue de aplicaciones como paquetes. Se proporciona un paquete OSGi como parte de IBM WebSphere MQ classes for Java .

OSGi proporciona una infraestructura Java con fines generales, segura y gestionada, que soporta el despliegue de aplicaciones en forma 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.

IBM WebSphere MQ classes for Java. incluye el siguiente paquete OSGi.

com.ibm.mq.osgi.java_< número de versión > .jar
Los archivos JAR permiten que las aplicaciones utilicen las IBM WebSphere MQ classes for Java.

donde < número de versión > es el número de versión de WebSphere MQ que se ha instalado.

El paquete se instala en el subdirectorio `java/lib/OSGi` de la instalación de IBM WebSphere MQ o en la carpeta `java\lib\OSGi` en Windows.

Otros nueve paquetes también se instalan en el subdirectorio `java/lib/OSGi` de la instalación de IBM WebSphere MQ o en la carpeta `java\lib\OSGi` en Windows. Estos paquetes forman parte de las IBM WebSphere 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 WebSphere MQ classes for Java. Si el paquete OSGi de IBM WebSphere MQ classes for Java se carga en un entorno de ejecución OSGi que también tiene los paquetes de IBM WebSphere MQ classes for JMS cargados, errores como:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

cuando se ejecutan aplicaciones que utilizan el paquete de IBM WebSphere MQ classes for Java o los paquetes de IBM WebSphere MQ classes for JMS .

El paquete OSGi para las IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java pero el paquete de las IBM WebSphere MQ classes for Java no reconoce el paquete de usuario. Como resultado, el cargador de clases utilizado en un paquete de IBM WebSphere MQ classes for Java no puede cargar una clase de salida de canal que esté en un paquete de usuario.

Para obtener más información sobre OSGi, consulte el sitio web de OSGi Alliance.

El archivo de configuración de IBM WebSphere MQ classes for Java

Un archivo de configuración de IBM WebSphere MQ classes for Java especifica las propiedades que se utilizan para configurar el IBM WebSphere MQ classes for Java.

Un archivo de configuración de IBM WebSphere MQ classes for Java tiene el formato de un archivo de propiedades estándar de Java.

V 7.5.0.9 A partir de IBM WebSphere MQ Version 7.5.0, Fixpack 9, se proporciona un archivo de configuración de ejemplo denominado `mqjava.config` en el subdirectorio `bin` del directorio de instalación de IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 de IBM WebSphere MQ classes for Java como el archivo `D:\mydir\mqjava.config` en el sistema Windows local.

Un archivo de configuración de IBM WebSphere 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 WebSphere MQ classes for Java

Un archivo de configuración de IBM WebSphere MQ MQI client también puede especificar las propiedades que se utilizan para configurar IBM WebSphere MQ classes for Java. Pero las propiedades que se especifican en un archivo de configuración de IBM WebSphere 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 WebSphere MQ MQI client especificando el atributo como propiedad en un archivo de configuración de IBM WebSphere MQ classes for Java. Para alterar temporalmente un atributo en un archivo de configuración de IBM WebSphere MQ MQI client, utilice una entrada con el formato siguiente en el archivo de configuración de IBM WebSphere 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 WebSphere MQ MQI client donde reside el atributo.

propName
El nombre del atributo tal como está especificado en el archivo de configuración de IBM WebSphere MQ MQI client.

propValue
El valor de la propiedad que altera temporalmente el valor del atributo especificado en el archivo de configuración de IBM WebSphere MQ MQI client.

Como alternativa, puede alterar temporalmente un atributo contenido en un archivo de configuración de IBM WebSphere 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 de IBM WebSphere MQ MQI client son relevantes para IBM WebSphere MQ classes for Java. Si especifica o altera temporalmente otros atributos, dicha acción no tendrá afecto. En concreto, tenga en cuenta que `ChannelDefinitionFile` y `ChannelDefinitionDirectory` en la stanza CHANNELS del archivo de configuración de cliente no se utilizan. Consulte “Utilización de una tabla de definiciones de canal de cliente con IBM WebSphere MQ classes for Java” en la página 686 para conocer detalles sobre cómo utilizar la tabla de definición de canal de cliente (CCDT) con IBM WebSphere MQ classes for Java.

Tabla 86. Stanzas del archivo de configuración de cliente y los atributos que contienen

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

Para obtener más información sobre la configuración de IBM WebSphere MQ MQI client, consulte Configuración de un cliente utilizando un archivo de configuración.

Tareas relacionadas
Rastreo de aplicaciones de IBM WebSphere MQ classes for Java

Stanza Standard Environment Trace de Java
Puede utilizar la stanza Java Standard Environment Trace Settings para configurar el recurso de rastreo de IBM WebSphere MQ classes for Java .

com.ibm.msg.client.commonservices.trace.outputName = traceOutputName
traceOutputName es el directorio y el nombre de archivo a los que se envía la salida de rastreo.

El nombre predeterminado del archivo de rastreo depende de la versión del IBM WebSphere MQ classes for Java que está utilizando una aplicación:

- Para IBM WebSphere MQ classes for Java para Version 7.5.0, Fix Pack 8 o anterior, *traceOutputName* toma como valor predeterminado un archivo denominado `mqjms_%PID%.trc` en el directorio de trabajo actual.
- V 7.5.0.9 A partir de IBM WebSphere MQ classes for Java de Version 7.5.0, Fix Pack 9, *traceOutputName* toma como valor predeterminado un archivo denominado `mqjava_%PID%.trc` en el directorio de trabajo actual.

donde *%PID%* es el ID de proceso actual. 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 paquetes y clases que se rastrean, o los valores especiales `ALL` o `NONE`.

Separe los nombres de paquete o clase con un punto y coma (`;`). **includeList** toma como valor predeterminado `ALL`y rastrea todos los paquetes y clases en IBM WebSphere 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 o clase con un punto y coma (`;`). **excludeList** toma como valor predeterminado `NONE`y, por lo tanto, no excluye ningún paquete ni clase de IBM WebSphere MQ classes for Java que se pueda rastrear.

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 desde cualquier matriz de bytes.

Si **maxArrayBytes** se establece en un entero positivo, limita el número de bytes de una matriz de bytes que se graban en el archivo de rastreo. Trunca la matriz de bytes después de escribir *maxArrayBytes* . El establecimiento de **maxArrayBytes** reduce el tamaño del archivo de rastreo resultante y reduce el efecto del rastreo en 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 graban 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 salida de rastreo a 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 tiene el sufijo .trc.0, el siguiente archivo de salida de rastreo más reciente finaliza con .trc.1, y así sucesivamente. 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 los valores de retorno se incluyen en el rastreo.

traceParameters toma como valor predeterminado TRUE. Si **traceParameters** se establece en FALSE, sólo se rastrean las firmas de método.

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 especifica en **traceOutputName** , se utiliza ese nombre en su lugar y 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 WebSphere MQ classes for Java que el rastreo no comprimido.

Si se establecen **maxTraceBytes** y **traceCycles** , se crean varios archivos de rastreo comprimidos en lugar de varios archivos sin formato.

Si IBM WebSphere MQ classes for Java 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 el IBM WebSphere MQ classes for Java se cierra de forma controlada. Utilice la compresión de rastreo sólo si los problemas que se están investigando no hacen que la propia JVM se detenga de forma inesperada. No utilice la compresión de rastreo cuando diagnostique problemas que puedan dar como resultado cierres de System.Halt() o terminaciones de JVM anormales y no controladas.

com.ibm.msg.client.commonservices.trace.level = traceLevel
traceLevel especifica un nivel de filtrado para el rastreo. Los niveles de rastreo definidos son los siguientes:

Tabla 87. Qué se rastrea para cada nivel de rastreo

Valor	Qué se rastrea
0	El rastreo está desactivado
1	Excepciones

Tabla 87. Qué se rastrea para cada nivel de rastreo (continuación)	
Valor	**Qué se rastrea**
3	Excepciones Avisos
6	Excepciones Avisos Puntos de rastreo informativos
8	Excepciones Avisos Puntos de rastreo informativos Entrada y salida de método
9	Excepciones Avisos Puntos de rastreo informativos Entrada y salida de método Datos que se envían entre IBM WebSphere MQ classes for Java y un gestor de colas.

Nota: Utilice siempre el valor 9 a menos que el soporte de IBM indique lo contrario.

IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java constan de una serie de archivos JAR. Cuando está desarrollando aplicaciones de lenguaje Java utilizando esta API, es necesaria una instalación de IBM WebSphere MQ Server, IBM WebSphere MQ Client o IBM WebSphere 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 WebSphere 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 WebSphere MQ.
- El usuario es responsable de actualizar el repositorio con cualquier mantenimiento proporcionado por el soporte de IBM .

Para IBM WebSphere MQ Version 7.5, es necesario instalar los siguientes archivos JAR en el repositorio:

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar

- com.ibm.mq.headers.jar
- connector.jar

Configuración posterior a la instalación para aplicaciones de IBM WebSphere MQ

Después de instalar IBM WebSphere MQ, puede configurar la instalación de modo que ejecute sus propias aplicaciones.

Recuerde que debe comprobar el archivo README de IBM WebSphere MQ para obtener información posterior o más específica para su entorno.

Antes de intentar ejecutar una aplicación IBM WebSphere MQ classes for Java en modalidad de enlaces, asegúrese de que ha configurado IBM WebSphere MQ tal como se describe en Configuración .

Configuración del gestor de colas para aceptar conexiones de cliente de clases WebSphere MQ para 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.

Para obtener más detalles, consulte “Preparación y ejecución de los programas de ejemplo” en la página 112.

Ejecución de clases de WebSphere MQ para aplicaciones Java bajo el Gestor de seguridad Java

Las clases de WebSphere MQ para Java se pueden ejecutar con el gestor de seguridad de Java habilitado. Para ejecutar aplicaciones correctamente con el gestor de seguridad habilitado, debe configurar la máquina virtual Java (JVM) con un archivo de definición de política adecuado.

La forma más sencilla de hacerlo es cambiar el archivo de políticas proporcionado con el JRE. En la mayoría de los sistemas, este archivo se almacena en la vía de acceso `lib/security/java.policy`, relativa al directorio JRE. Puede editar archivos de políticas utilizando el editor que prefiera o el programa `policytool` proporcionado 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 las clases de WebSphere MQ para Java cuando se ejecuta bajo el Gestor de seguridad de Java.

A continuación se muestra un ejemplo de una entrada de archivo de políticas que permite que las clases de WebSphere MQ para Java se ejecuten correctamente bajo el gestor de seguridad predeterminado:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
```

```
"com.ibm.vm.bitmode","read";
};
grant codeBase
"file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.commonservices.jar" {
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.mq.commonservices", "read";
  permission java.util.logging.LoggingPermission "control";
  //For access to the trace properties file.
  permission java.io.FilePermission "/tmp/trace.properties", "read";
  //For access to the trace output files.
  permission java.io.FilePermission "/tmp/*", "read,write";
};
```

Notas:

- *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ.
- Este ejemplo de un archivo de políticas permite que las clases WebSphere MQ para Java funcionen correctamente bajo el gestor de seguridad, pero es posible que tenga que habilitar su propio código para que se ejecute correctamente antes de que funcionen las aplicaciones.
- Para permitir que WebSphere MQ classes for Java acceda a los archivos JAR (Java archive) de una aplicación, añada el siguiente permiso a la primera sentencia `grant` :

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- Para utilizar estas sentencias `grant` en el archivo de configuración de políticas, es posible que tenga que modificar los nombres de vía de acceso en función de dónde haya instalado las clases de WebSphere MQ para Java y dónde almacene las aplicaciones.
- El código de ejemplo suministrado con WebSphere MQ classes for Java no se ha habilitado específicamente para su uso con el gestor de seguridad; sin embargo, las pruebas IVT se ejecutan con este archivo de política y el gestor de seguridad predeterminado en su lugar.

Verificación de las clases IBM WebSphere MQ para la instalación de Java

Se proporciona un programa de verificación de instalación, MQIVP, con clases IBM WebSphere MQ para Java. Puede utilizar este programa para probar todas las modalidades de conexión de las clases IBM WebSphere MQ para 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 “Preparación y ejecución de los programas de ejemplo” en la página 112. 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 clases de WebSphere MQ para Java” en la página 666.

   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 relevantes para WebSphere MQ classes for Java” en la página 670.
4. Cambie el directorio a `MQ_INSTALLATION_PATH/mqm/VRM/java/samples/wmqjava`, donde MQ_INSTALLATION_PATH es la vía de acceso a la instalación de IBM WebSphere MQ y `VRM` es la versión, el release y el número de modificación del producto. En el indicador de mandatos, escriba:

   ```
   java -Djava.library.path=library_path MQIVP
   ```

   donde *vía_acceso_biblioteca* es la vía de acceso a las bibliotecas de IBM WebSphere MQ classes for Java (consulte Las clases de WebSphere MQ para bibliotecas Java).

En el mensaje de solicitud marcado (1):

- Para utilizar una conexión TCP/IP, especifique un nombre de host de servidor IBM WebSphere 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 WebSphere 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. Si elige la conexión de servidor, no verá los mensajes de solicitud marcados (2).

Resolución de problemas de IBM WebSphere MQ

Inicialmente, ejecute el programa de verificación de la instalación. Puede también ser necesario utilizar el recurso de rastreo.

Si un programa no finaliza correctamente, ejecute el programa de verificación de la instalación y siga los consejos que se proporcionan en los mensajes de diagnóstico. Este programa se describe en “Verificación de las clases IBM WebSphere MQ para la instalación de Java” en la página 679.

Si los problemas continúan y necesita ponerse en contacto con el equipo de servicio de IBM , es posible que se le solicite 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).
- Entre el siguiente mandato:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path=library_path MQIVP -trace
```

donde:

- *archivo_propiedades_commonservices* es la vía de acceso (incluido el nombre de archivo) del archivo de propiedades *com.ibm.mq.commonservices*.
- *library_path* es la vía de acceso a las clases de WebSphere MQ para bibliotecas Java (consulte Clases de WebSphere MQ para bibliotecas Java).

Para obtener más información sobre cómo utilizar el rastreo, consulte Rastreo de aplicaciones IBM WebSphere MQ classes for Java.

introducción para programadores

Esta colección de temas contiene información general para los programadores.

Para obtener información más detallada sobre cómo escribir programas, consulte “Escritura de clases de WebSphere MQ para aplicaciones Java” en la página 681.

Las clases de WebSphere MQ para la interfaz Java

La interfaz de programación de aplicaciones de WebSphere MQ utiliza verbos, que actúan sobre los objetos. La interfaz de programación Java utiliza objetos, sobre los que se actúa llamando a métodos.

La interfaz de programación de aplicaciones de WebSphere MQ de procedimiento se basa en verbos como los siguientes:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Todos estos verbos toman, como parámetro, un manejador para el objeto WebSphere MQ en el que van a operar. Debido a que Java está orientado a objetos, la interfaz de programación Java cambia esta ronda. El programa consta de un conjunto de objetos WebSphere MQ , sobre los que puede actuar llamando a métodos en dichos 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 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();
```

Escritura de clases de WebSphere MQ para aplicaciones Java

Esta colección de temas proporciona información para ayudarle a escribir aplicaciones Java para interactuar con sistemas WebSphere MQ .

Para utilizar WebSphere MQ classes for Java para acceder a las colas de WebSphere MQ , escriba aplicaciones Java que contengan llamadas que pongan mensajes y obtengan mensajes de las colas de WebSphere MQ . Para obtener detalles de las clases individuales, consulte WebSphere MQ classes for Java.

Nota: La reconexión automática de cliente no está soportada por las clases WebSphere MQ para Java.

WebSphere Clases MQ para modalidades de conexión Java

La forma en que programa para WebSphere MQ classes for Java tiene algunas dependencias en las modalidades de conexión que desea utilizar.

Si utiliza conexiones de cliente, hay varias diferencias respecto del IBM WebSphere 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.

WebSphere MQ Clases para conexiones de cliente Java

Cuando se utilizan clases de WebSphere MQ para Java como cliente, es como el IBM WebSphere MQ MQI client, pero tiene una serie de diferencias.

Si está programando para *WebSphere MQ classes for Java* para su uso como cliente, tenga en cuenta las diferencias siguientes:

- Sólo ofrece soporte para TCP/IP.
- No lee ninguna variable de entorno de WebSphere MQ durante el 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 de error predeterminado es la consola Java.
- Sólo los atributos siguientes en un archivo de configuración de cliente de WebSphere MQ son relevantes para las clases de WebSphere MQ para 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	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

- Si se conecta a un gestor de colas que requiere la conversión de datos de tipo carácter, el cliente Java V7 ahora es capaz de realizar la conversión si el gestor de colas no puede hacerlo. La JVM de cliente debe permitir la conversión entre el CCSID del cliente y el CCSID del gestor de colas.
- La reconexión automática del cliente no está soportada en WebSphere MQ classes for Java.

Cuando se utiliza en modalidad de cliente, las clases *WebSphere MQ para Java* no dan soporte a la llamada MQBEGIN.

Consulte “Opciones de conexión para clases de WebSphere MQ para Java” en la página 667 para obtener más información sobre los entornos soportados.

WebSphere MQ Clases para la modalidad de enlaces Java

La modalidad de enlaces de WebSphere MQ classes for Java difiere de la modalidad de cliente en tres formas principales.

Cuando se utiliza en modalidad de enlaces, WebSphere MQ classes for Java utiliza la interfaz nativa Java (JNI) para llamar directamente a la API del gestor de colas existente, en lugar de comunicarse a través de una red.

De forma predeterminada, las aplicaciones que utilizan las clases WebSphere MQ para Java en modalidad de enlaces se conectan a un gestor de colas utilizando *ConnectOption*, MQCNO_STANDARD_BINDINGS.

Las clases de WebSphere MQ para Java dan soporte a las siguientes *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 213.

La modalidad de enlaces da soporte a la llamada MQBEGIN para iniciar unidades de trabajo globales coordinadas por el gestor de colas, en todas las plataformas excepto WebSphere MQ para IBM i y WebSphere MQ para 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.

Consulte "Opciones de conexión para clases de WebSphere MQ para Java" en la página 667 para obtener más información sobre los entornos soportados.

Definición de las clases de WebSphere MQ para la conexión Java que se va a 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 WebSphere 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 conectarse y desconectarse de un gestor de colas utilizando WebSphere MQ classes for Java.

Configuración del entorno de WebSphere MQ para WebSphere 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. *No* es relevante si se conecta en modalidad de enlaces. Consulte: "Modalidades de conexión para WebSphere MQ classes for JMS" en la página 789

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 Java intentan conectarse a un escucha de WebSphere 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 WebSphere MQ Explorer **Conexiones de aplicaciones** (donde el campo se denomina **App name**).

Los nombres de aplicación están limitados a 28 caracteres y los nombres más largos se truncan para que quepan. 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 WebSphere 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`.

Nota: Los gestores de colas que se ejecutan en plataformas z/OS no dan soporte al establecimiento de nombres de aplicación.

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 WebSphere MQ

Un archivo de configuración de cliente de WebSphere MQ también puede especificar propiedades que se utilizan para configurar WebSphere MQ classes for Java. Sin embargo, las propiedades especificadas en un archivo de configuración de cliente MQI de WebSphere MQ sólo se aplican cuando una aplicación se conecta a un gestor de colas en modalidad de cliente.

Si es necesario, puede alterar temporalmente cualquier atributo de un archivo de configuración de WebSphere 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 Java5 y releases posteriores, establezca una variable de entorno del sistema.

Sólo los atributos siguientes en un archivo de configuración de cliente de WebSphere MQ son relevantes para las clases de WebSphere MQ para 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 clases de WebSphere MQ para 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 WebSphere MQ classes for Java

Una aplicación cliente de IBM WebSphere MQ classes for Java puede utilizar definiciones de canal de conexión de cliente almacenadas en una tabla de definiciones 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 WebSphere MQ classes for Java puede utilizar definiciones de canal de conexión de cliente almacenadas en una tabla de definiciones de canal de cliente. Estas definiciones se crean mediante los mandatos IBM WebSphere MQ Script (MQSC) o IBM WebSphere MQ Programmable Command Format (PCF), o utilizando IBM WebSphere MQ Explorer.

Cuando la aplicación crea un objeto `MQQueueManager` , el cliente IBM WebSphere MQ classes for Java busca en la tabla de definición de canal de cliente una definición de canal de conexión de cliente adecuada y utiliza la definición de canal 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 las clases IBM WebSphere MQ classes for Java accedan a la tabla de definición de canal de cliente identificada por el objeto de URL chanTab2, busquen en la tabla una definición de canal de conexión de cliente adecuada y, a continuación, utilicen la definición de canal para iniciar un canal MQI en el gestor de colas denominado 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 WebSphere MQ classes for Java emite un `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 199.

  Si la aplicación utiliza la agrupación de conexiones, consulte "Control de la agrupación de conexiones predeterminada en las clases de WebSphere MQ para Java" en la página 707.
- Cuando el cliente de IBM WebSphere MQ classes for Java encuentra una definición de canal de conexión de cliente adecuada en la tabla de definiciones 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 concreto, tenga en cuenta los puntos siguientes si utiliza SSL (Secure Sockets Layer):

  - Un canal MQI utiliza SSL sólo si la definición de canal extraída de la tabla de definición de canal de cliente especifica el nombre de una CipherSpec soportada por las clases IBM WebSphere 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). Las clases IBM WebSphere MQ classes for Java sólo utilizan esta información para acceder a los servidores LDAP que contienen las CRL.
  - Una tabla de definiciones de canal de cliente también puede contener la ubicación de un programa de respuesta OCSP (Online Certificate Status Protocol). Las clases IBM WebSphere MQ classes for Java no pueden utilizar la información OCSP en un archivo de tabla de definición de canal de cliente. Sin embargo, puede configurar OCSP como se describe en la sección Utilización del protocolo de certificados en línea.

  Para obtener más información sobre cómo utilizar SSL con una tabla de definiciones de canal de cliente, consulte Especificación de que un canal MQI utiliza SSL.

  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 clases de WebSphere MQ para Java" en la página 700. 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 clases WebSphere MQ para Java" en la página 704.

Especificación de un rango de puertos para las conexiones de cliente de IBM WebSphere 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 IBM WebSphere MQ classes for Java intenta conectarse a un gestor de colas IBM WebSphere MQ en modalidad de cliente, un cortafuegos puede permitir sólo las conexiones que se originan en puertos o rangos de puertos especificados. 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 WebSphere 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 clases de WebSphere MQ para 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 de estos, consulte Clase 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 clases WebSphere MQ para 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 procedimiento, donde MQPUT y MQGET colocan y obtienen matrices de bytes, el lenguaje de programación Java coloca 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 nueva instancia de mensaje, todos los parámetros MQMD se establecen automáticamente en sus valores predeterminados, tal como se define en Valores iniciales y declaraciones de idioma 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 de número que utilizan los métodos de lectura y escritura estableciendo la variable de miembro *encoding* .

Puede modificar el juego de caracteres para que se utilice para leer y escribir series estableciendo la variable de miembro *characterSet* .

Para obtener más información, consulte “Clase MQMessage” en la página 1089.

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 Java (utilizando readUTF()), esta es la forma más sencilla de enviar información de serie.

Mejora del rendimiento de los mensajes no persistentes en las clases WebSphere MQ para 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 siguiente.

En las clases WebSphere MQ para Java, utilice 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 navegadores de colas pueden utilizar la lectura anticipada en ese objeto.

Transferencia asíncrona de mensajes utilizando clases WebSphere MQ para 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 clases de WebSphere MQ para Java

En las clases de WebSphere MQ para Java, el tema se representa mediante la clase MQTopic y se publica en él utilizando los métodos MQTopic.put().

Para obtener información general sobre la publicación/suscripción de WebSphere MQ , consulte Introducción a WebSphere MQ mensajería de publicación/suscripción .

Manejo de cabeceras de mensajes de WebSphere MQ con clases WebSphere MQ para Java

Se proporcionan clases Java que representan distintos tipos de cabecera de mensaje. También se proporcionan dos clases auxiliares.

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.

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 cabecera, pero también le permite buscar cabeceras concretas, añadir nuevas cabeceras, eliminar cabeceras existentes, actualizar campos de cabecera y, a continuación, volver a escribir el contenido de 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 WebSphere MQ están asociadas a tipos y formatos de mensaje concretos. El MQHeaderRegistry se configura con conocimiento de todos los formatos y tipos de cabecera actuales de WebSphere MQ y sus clases de implementación, y también puede registrar sus propios tipos de cabecera.

Se proporciona soporte para las siguientes cabeceras de Websphere MQ utilizadas habitualmente

- MQRFH - Cabecera de reglas y formato
- MQRFH2 -Al igual que MQRFH, se utiliza para pasar mensajes a y desde un intermediario de mensajes que pertenece a WebSphere Message Broker. También se utiliza para contener propiedades de mensaje
- MQCIH-Puente CICS
- MQDLH - Cabecera de mensajes no entregados
- MQIIH-Cabecera de 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.

Impresión de todas las cabeceras en un mensaje utilizando clases WebSphere MQ para 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 utilizando clases WebSphere MQ para 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 de mensaje no enviado utilizando las clases WebSphere MQ para 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 MQDLH de un mensaje de mensaje no entregado utilizando WebSphere 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.
```

Impresión del contenido de un mensaje utilizando clases WebSphere MQ para 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 utilizando clases WebSphere MQ para 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 utilizando clases WebSphere MQ para 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 con nombre-en esta instancia, el campo Rto en la carpeta jms, que representa el nombre de cola de respuestas en un mensaje JMS de MQ .

```
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 corrientes de bytes distintas de los objetos MQMessage utilizando clases WebSphere MQ para Java

Estos ejemplos utilizan las clases de cabecera para analizar y manipular el contenido de la cabecera WebSphere MQ cuando el origen de datos no es un objeto MQMessage.

Puede utilizar las clases de cabecera para analizar y manipular el contenido de cabecera de WebSphere MQ incluso cuando el origen de datos es distinto de 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 corrientes de datos, por ejemplo, corrientes de archivos o sockets, o matrices de bytes transportadas en mensajes JMS. Las clases java.io.DataInputStream implementan DataInput, y las clases java.io.DataOutputStream implementan DataOutput. En este ejemplo se lee el contenido de la cabecera WebSphere MQ 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 utilizando clases WebSphere MQ para Java

Puede crear clases Java para tipos de cabecera no suministrados con clases WebSphere MQ para 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 WebSphere MQ classes for Java, debe crear 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 clases WebSphere MQ para Java

Se proporcionan clases Java para crear y analizar mensajes estructurados por PCF, y para 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 mensajes en clases WebSphere MQ para Java

Las llamadas de función para procesar manejadores de mensajes no tienen ningún equivalente en las clases WebSphere MQ para 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 19.

En WebSphere MQ , el acceso de Java a los mensajes se realiza a través de la clase MQMessage. Por lo tanto, los manejadores de mensajes no se proporcionan en el entorno Java y no hay ningún equivalente a las llamadas de función WebSphere MQ MQCRTMH, MQDLTMH, MQMHBUF y MQBUFMH

Para establecer propiedades de descriptor de contexto de mensaje en la interfaz de procedimientos, utilice la llamada MQSETMP. En WebSphere MQ classes for Java, utilice el método adecuado 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 WebSphere MQ classes for Java, utilice el método adecuado 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 WebSphere MQ classes for Java, utilice el método deleteProperty de la clase MQMessage.

Manejo de errores en clases de WebSphere MQ para Java

Maneje los errores que surgen de las clases WebSphere MQ para Java utilizando bloques Java `try` y `catch` .

Los métodos de la interfaz Java no devuelven un código de terminación y un código de razón. En su lugar, emiten una excepción siempre que el código de terminación y el código de razón resultantes de una llamada WebSphere MQ no son ambos cero. Esto simplifica la lógica del programa para que no tenga que comprobar los códigos de retorno después de cada llamada a WebSphere 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 WebSphere MQ notificados en excepciones Java para z/OS se documentan en Códigos de razón para z/OS y Códigos de razón para todas las demás plataformas.

Las excepciones que se emiten mientras se ejecuta una aplicación WebSphere MQ para Java también se graban en el 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.

De forma predeterminada, la clase MQException transmite automáticamente las excepciones a System.err, que normalmente se dirige a la consola. Si desea detener las excepciones que aparecen en la consola, incluya una línea en la aplicación para establecer MQException.log= null.

Obtención y establecimiento de valores de atributo en clases WebSphere MQ para 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 *nuevo* , se abre automáticamente para su consulta. Cuando utiliza el método accessProcess() para acceder a un objeto de proceso, el objeto se abre automáticamente para inquire. Cuando se utiliza el método accessQueue() para acceder a un objeto de cola, dicho 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 en 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 inherentemente multihebra. WebSphere MQ classes for Java permite que varias hebras compartan un objeto de gestor de colas, 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 inherentemente multihebra. 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 cliente MQI de WebSphere MQ basado en C, esto provocaría un problema, porque existen limitaciones en la compartición de manejadores por parte de varias hebras. WebSphere MQ classes for Java relaja esta restricción, permitiendo que varias hebras compartan un objeto de gestor de colas (y sus objetos de cola, tema y proceso asociados).

La implementación de WebSphere MQ classes for Java garantiza que, para una conexión determinada (instancia de objetoMQQueueManager), se sincronice todo el acceso al gestor de colas WebSphere MQ de destino. 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 clases WebSphere MQ para Java

Una visión general de cómo utilizar salidas de canal en una aplicación utilizando las clases de WebSphere MQ para Java.

Los temas siguientes describen cómo escribir una salida de canal en Java, cómo asignarla y cómo pasarle datos. 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 clases de WebSphere MQ para 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 las clases WebSphere MQ para Java, por ejemplo, si está utilizando una aplicación cliente escrita en C.

Cualquier cifrado SSL definido para una conexión se realiza *después* de que se hayan invocado las salidas de envío y 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 WebSphere 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 WebSphere MQ anteriores a 7.0, implementaría estas salidas utilizando las interfaces MQSendExit, MQReceiveExity 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 WebSphere MQ classes for Java

Puede asignar una salida de canal utilizando IBM WebSphere MQ classes for Java.

No hay ningún equivalente directo al canal de IBM WebSphere MQ en IBM WebSphere 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 de IBM WebSphere MQ anteriores a la versión 7.0, y el campo de salida channelSend, que se puede utilizar para cualquier salida de envío), 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 WebSphere 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 88 en la página 703 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 88 en la página 703 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 88. Directorio para los programas de salida de canal

Plataforma	**Directorio**
AIX, HP-UX, Linuxy Solaris	`/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 WebSphere MQ durante la instalación. El directorio predeterminado es `C:\Program Files\IBM\WebSphere MQ`.	

Pasar datos a salidas de canal en clases WebSphere MQ para 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 WebSphere 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 WebSphere MQ classes for Java” en la página 686.

Utilización de salidas de canal no escritas en Java con clases WebSphere MQ para Java

Cómo utilizar programas de salida de canal escritos en C desde una aplicación Java.

En WebSphere MQ Versión 7.0, puede especificar el nombre de un programa de salida de canal escrito en C como una serie pasada a los campos Salida channelSecurity, Salida channelSendo Salida channelReceiveen el objeto MQEnvironment o propiedades Hashtable. Sin embargo, no puede utilizar una salida de canal escrita en Java en una aplicación escrita en otro lenguaje.

Especifique el nombre del programa de salida con el formato `library(function)` y asegúrese de que la ubicación del programa de salida se incluye en la variable de entorno de vía de acceso.

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 405.

Utilización de clases de salida externas

En las versiones de WebSphere MQ anteriores a la versión 7.0, se proporcionaron tres clases para permitirle utilizar salidas de canal escritas en lenguajes distintos de Java:

- MQExternalSecurityExit, que implementa la interfaz MQSecurityExit
- MQExternalSendExit, que implementa la interfaz MQSendExit
- MQExternalReceiveExit, que implementa la interfaz MQReceiveExit

El uso de estas clases sigue siendo válido, pero es preferible el nuevo método.

Para utilizar una salida de seguridad que no está escrita en Java, primero una aplicación tuvo que crear un objeto de salida MQExternalSecurity. La aplicación especifica, como parámetros en el constructor MQExternalSecurityExit, el nombre de la biblioteca que contiene la salida de seguridad, el nombre del punto de entrada para la salida de seguridad y los datos de usuario que se deben pasar a la salida de seguridad cuando se invoque. Los programas de salida de canal que no están escritos en Java se han almacenado en el directorio que se muestra en la Tabla 88 en la página 703.

Utilización de una secuencia de salidas de envío o recepción de canal en clases WebSphere MQ para Java

Una aplicación WebSphere MQ para Java puede utilizar una secuencia de salidas de envío o recepción de canal que se ejecutan de forma sucesiva.

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 envío escrita en Java)
- Una instancia de la clase de salida MQExternalSend(para una salida de envío 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 en la salida C.

Utilización de clases de cadena de salidas

En las versiones de WebSphere MQ anteriores a la versión 7.0, se proporcionaron 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. El uso de las interfaces de WebSphere MQ Classes for Java significa que la aplicación sigue teniendo una dependencia en `com.ibm.mq.jar` Si se utiliza el nuevo conjunto de interfaces en el paquete `com.ibm.mq.exits` , no hay ninguna dependencia en `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 envío escrita en Java)
- Una instancia de la clase de salida MQExternalSend(para una salida de envío 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 clases de WebSphere MQ para 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 WebSphere MQ classes for Java utilice la función de compresión integrada en IBM WebSphere MQ.

Esta función proporcionada con IBM WebSphere 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 WebSphere MQ, consulte Compresión de datos (COMPMSG)y Compresión de cabecera (COMPHDR).

Una aplicación IBM WebSphere 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.

Para obtener más información sobre cómo especificar técnicas de compresión y sobre qué técnicas de compresión están disponibles, consulte Clase com.ibm.mq.MQEnvironment e Interfaz com.ibm.mq.MQC .

Compartimiento de una conexión TCP/IP en IBM WebSphere MQ classes for Java

Se pueden crear varias instancias de un canal MQI para que compartan una sola conexión TCP/IP.

En IBM WebSphere 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 WebSphere MQ classes for Java, no es seguro que una nueva solicitud de conexión compartirá siempre una conexión ya establecida.

Agrupación de conexiones en clases WebSphere MQ para Java

WebSphere MQ classes for Java permite agrupar las conexiones de repuesto para reutilizarlas.

WebSphere MQ classes for Java proporciona soporte adicional para aplicaciones que se ocupan de varias conexiones con gestores de colas de WebSphere 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.

WebSphere 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 WebSphere MQ classes for Java construye un objeto MQQueueManager , 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 las clases de WebSphere MQ para 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.

WebSphere MQ classes for Java proporciona una agrupación de conexiones predeterminada que puede utilizar para hacerlo. 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 diferente en las clases WebSphere MQ para Java” en la página 709).

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 las clases WebSphere MQ para 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 diferente en las clases WebSphere MQ para 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 WebSphere MQ de forma independiente. Estas hebras utilizan myConnMan cuando crean conexiones.

Suministro de sus propias clases ConnectionManager para WebSphere MQ para Java

WebSphere MQ classes for Java proporciona una implementación parcial de Java EE Connector Architecture, lo que permite utilizar implementaciones de javax.resource.spi.ConnectionManager .

Las aplicaciones y los proveedores de middleware pueden proporcionar implementaciones alternativas de agrupaciones de conexiones. WebSphere MQ classes for Java proporciona una implementación parcial de Java EE Connector Architecture. Las implementaciones de **javax.resource.spi.ConnectionManager** pueden utilizarse como Connection Manager predeterminado o especificarse en el constructor MQQueueManager .

WebSphere MQ classes for Java cumple con el contrato de gestión de conexiones de Java EE Connector Architecture. Lea esta sección junto con el contrato de gestión de conexiones de la arquitectura de conector Java EE (consulte el sitio web Java de Sun en https://java.sun.com).

La interfaz ConnectionManager sólo define un método:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

El constructor MQQueueManager llama a allocateConnection en el ConnectionManageradecuado. Pasa las implementaciones adecuadas de ManagedConnectionFactory y ConnectionRequestInfo como parámetros para describir la conexión necesaria.

El ConnectionManager busca en su agrupación un objeto javax.resource.spi.ManagedConnection que se ha creado con objetos ManagedConnectionFactory y ConnectionRequestInfo idénticos. Si el ConnectionManager encuentra objetos ManagedConnection adecuados, crea un java.util.Set que contiene el candidato ManagedConnections. A continuación, el ConnectionManager llama a lo siguiente:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

La implementación de WebSphere MQ de ManagedConnectionFactory ignora el parámetro subject. Este método selecciona y devuelve una ManagedConnection adecuada del conjunto, o devuelve un valor nulo si no encuentra una ManagedConnectionadecuada. Si no hay una ManagedConnection adecuada en la agrupación, el ConnectionManager puede crear una utilizando:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

De nuevo, el parámetro subject se ignora. Este método se conecta a un gestor de colas de WebSphere MQ y devuelve una implementación de javax.resource.spi.ManagedConnection que representa la conexión recién forjada. Una vez que ConnectionManager ha obtenido una ManagedConnection (de la agrupación o recién creada), crea un descriptor de conexión utilizando:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

Este descriptor de conexión se puede devolver desde allocateConnection().

Un ConnectionManager debe registrar un interés en ManagedConnection a través de:

```
mc.addConnectionEventListener()
```

Se notifica al escucha ConnectionEventsi se produce un error grave en la conexión, o cuando se llama a MQQueueManager.disconnect (). Cuando se llama a MQQueueManager.disconnect (), el escucha ConnectionEventpuede realizar una de las acciones siguientes:

- Restablezca ManagedConnection utilizando la llamada mc.cleanup() y, a continuación, devuelva ManagedConnection a la agrupación
- Destruya ManagedConnection utilizando la llamada mc.destroy()

Si ConnectionManager es el ConnectionManagerpredeterminado, también puede registrar un interés en el estado del conjunto gestionado por MQEnvironment de MQPoolTokens. Para ello, primero construya un objeto MQPoolServices y, a continuación, registre un objeto MQPoolServicesEventListener con el objeto MQPoolServices :

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

Se notifica al escucha cuando se añade o elimina una MQPoolToken del conjunto, o cuando cambia el ConnectionManager predeterminado. El objeto MQPoolServices también proporciona una forma de consultar el tamaño actual del conjunto de MQPoolTokens.

Coordinación JTA/JDBC utilizando clases WebSphere MQ para Java

WebSphere MQ classes for Java da soporte al método MQQueueManager.begin (), que permite a WebSphere MQ actuar como coordinador para una base de datos que proporciona un controlador compatible con 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 https://www.ibm.com/software/integration/wmq/requirements/.

Para utilizar el soporte de XA-JTA, debe utilizar la biblioteca de conmutadores especial de JTA. El método para utilizar esta biblioteca varía en función de si está utilizando Windows o una de las otras plataformas.

Configuración de la coordinación JTA/JDBC en Windows

La biblioteca XA se suministra como una DLL cuyo nombre tiene el formato `jdbcxxx.dll`.

V 7.5.0.7 El `jdbcora12.dll` proporcionado proporciona compatibilidad con Oracle 12C, para una instalación de servidor de IBM WebSphere MQ 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 se encuentra en el directorio `java\lib\jdbc` o `java\lib64\jdbc` de las clases IBM WebSphere MQ para la instalación de Java . Debe declarar la biblioteca XA, también descrita como archivo de carga conmutada, en el gestor de colas. Utilice IBM WebSphere 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`

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, WebSphere 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 proporcionado con WebSphere MQ. Una biblioteca de conmutadores necesita otras bibliotecas, que pueden estar almacenadas en diversas ubicaciones en sistemas diferentes. Pero una biblioteca de conmutadores no puede utilizar la variable de entorno LIBPATH para localizar estas bibliotecas, pues el encargado de cargar la biblioteca de conmutadores es el gestor de colas, el cual 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 WebSphere MQ . Para crear una biblioteca de conmutadores de 64 bits, emita 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.

Esto es un ejemplo de un mandato **make**:

```
make db2
```

Tenga en cuenta las siguientes cuestiones:

- 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 conmutador es jdbcdb2 y, para Oracle, el nombre de cada biblioteca de conmutador es jdbcora. Los archivos make garantizan que las bibliotecas de conmutación de 32 bits y 64 bits se almacenen en distintos directorios de WebSphere 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.
- Debido a que puede instalar Oracle en cualquier lugar del sistema, los archivos make utilizan la variable de entorno ORACLE_HOME para localizar el lugar en el que está instalado Oracle.

Después de haber creado las bibliotecas de conmutación para DB2, Oracle, o ambas, 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 u Oracle , debe sustituir la entrada SwitchFile en cada stanza por una de las siguientes:

Para una base de datos DB2

```
SwitchFile=jdbcdb2
```

En el caso de 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 u Oracle , o si desea añadir stanzas XAResourceManager adicionales, consulte Administración 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 u 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

Hay determinados problemas y limitaciones del soporte de JTA/JDBC , algunos en función del sistema de gestión de bases de datos en uso.

Puesto que este soporte 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. **Siempre** evite cerrar abruptamente una base de datos mientras haya aplicaciones que tengan conexiones abiertas con ella.

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

A 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
```

Consulte la documentación de DB2 para obtener más información.

La stanza XAResourceManager se debe configurar para utilizar ThreadOfControl=PROCESS. Para DB2 versión 8.1 y posterior, esto no coincide con el valor de hebra de control predeterminado para DB2, por lo que se debe especificar toc=p en la serie XA Open. 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 Java que utilizan la coordinación JTA/JDBC sean multihebra ellas mismas.

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().

Soporte SSL (Secure Sockets Layer) en clases WebSphere MQ para Java

Las clases WebSphere MQ para aplicaciones cliente Java dan soporte al cifrado SSL (Secure Sockets Layer). Necesita un proveedor JSSE para utilizar el cifrado SSL.

Las clases WebSphere MQ para aplicaciones cliente Java que utilizan TRANSPORT (CLIENT) dan soporte al cifrado SSL (Secure Sockets Layer). SSL proporciona cifrado de comunicaciones, autenticación e integridad de mensajes. Se suele utilizar para proteger las comunicaciones entre dos interlocutores cualesquiera en Internet o dentro de una intranet.

WebSphere MQ classes for Java utiliza JSSE (Java Secure Socket Extension) para manejar el cifrado SSL y, por lo tanto, requiere 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 WebSphere MQ para Java utiliza una tabla de definición de canal de cliente (CCDT) para conectarse a un gestor de colas, consulte “Utilización de una tabla de definiciones de canal de cliente con IBM WebSphere MQ classes for Java” en la página 686.

Habilitación de SSL en IBM WebSphere MQ classes for Java

Para habilitar SSL, especifique una CipherSuite. Hay dos maneras de especificar una CipherSuite.

SSL sólo está soportado para conexiones de cliente. Para habilitar SSL, debe especificar la CipherSuite que se utilizará al comunicarse 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 “SSL CipherSpecs y CipherSuites en WebSphere MQ classes for Java” en la página 720 contiene una tabla que correlaciona las CipherSpecs soportadas por IBM WebSphere MQ con las CipherSuites equivalentes tal como las conoce JSSE.

Para habilitar SSL, especifique la CipherSuite utilizando la variable de miembro estático sslCipherSuite de MQEnvironment. El ejemplo siguiente se conecta a un canal SVRCONN denominado

SECURE.SVRCONN.CHANNEL, que se ha configurado para requerir SSL con una CipherSpec de RC4_MD5_EXPORT:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Aunque el canal tiene una CipherSpec de RC4_MD5_EXPORT, la aplicación Java debe especificar una CipherSuite de SSL_RSA_EXPORT_WITH_RC4_40_MD5. Consulte el “SSL CipherSpecs y CipherSuites en WebSphere MQ classes for Java” en la página 720 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 WebSphere MQ classes for Java” en la página 686

Si necesita una conexión de cliente para utilizar una CipherSuite soportada por el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS), una aplicación puede establecer el campo sslFipsRequired en 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 WebSphere 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 conectarse correctamente utilizando SSL, el almacén de confianza JSSE debe estar configurado con certificados raíz de entidad emisora de certificados desde los que 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 UNIX, Linux y Windows

Utilización del nombre distinguido del gestor de colas en IBM WebSphere MQ classes for Java

El gestor de colas se identifica a sí mismo utilizando un certificado SSL, que contiene un nombre distinguido (DN). Una aplicación cliente de IBM WebSphere MQ classes for Java puede utilizar este DN 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 WebSphere MQ classes for Java

Especifique las listas de revocación de certificados que se deben utilizar mediante java.security.cert.CertStore class.IBM WebSphere MQ classes for Java a continuación, comprueba los certificados en la CRL 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 WebSphere MQ, consulte Trabajar con listas de revocación de certificados y listas de revocación de autorizaciones y Acceso a las CRL y ARL con WebSphere MQ classes for Java y WebSphere 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 las clases WebSphere MQ para Java

Una aplicación cliente WebSphere MQ para Java puede controlar cuándo se renegocia la clave secreta que se utiliza para el cifrado en una conexión de cliente, en términos del 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 sslReseto la propiedad de entorno MQC.SSL_RESET_COUNT_PROPERTY representa el número total de bytes enviados y recibidos por las clases WebSphere MQ para el código de cliente 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 también incluye la información de control enviada y recibida por las clases de WebSphere MQ para el cliente 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 WebSphere 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 WebSphere MQ classes for Java utilice este SSLSocketFactory para conectarse al gestor de colas IBM WebSphere 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 el almacén de confianza de JSSE en las clases WebSphere MQ para 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 almacén de confianza JSSE, o cambia la ubicación del almacén de claves o archivo de almacén de confianza, las clases WebSphere MQ para aplicaciones Java que se ejecutan en ese momento no recogen 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 SSL con clases WebSphere MQ para Java

Las clases de WebSphere MQ para Java pueden emitir una serie de códigos de razón al conectarse a un gestor de colas utilizando SSL.

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 da soporte a SSL.

MQRC_JSSE_ERROR
El proveedor JSSE ha informado de un error que no ha podido manejar WebSphere MQ. 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 SSL especificadas, pero se ha producido un error durante la inicialización del entorno SSL.

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 WebSphere MQ en "SSL CipherSpecs y CipherSuites en WebSphere MQ classes for Java" en la página 720.

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.

SSL CipherSpecs y CipherSuites en WebSphere MQ classes for Java
Si una aplicación IBM WebSphere MQ classes for Java puede establecer una conexión con un gestor de colas depende de la CipherSpec especificada en el extremo del servidor del canal MQI y de la CipherSuite especificada en el extremo del cliente.

Para cada combinación de CipherSpec y CipherSuite, si una aplicación IBM WebSphere MQ classes for Java se puede conectar a un gestor de colas depende del valor del campo sslFipsObligatorio en la clase MQEnvironment, o del valor de la propiedad de entorno CMQC.SSL_FIPS_REQUIRED_PROPERTY.

En el extremo del servidor de un canal MQI, el nombre de una CipherSpec se puede especificar como el valor del parámetro SSLCIPH en un mandato DEFINE CHANNEL CHLTYPE (SVRCONN). En el extremo del cliente de un canal MQI, una aplicación IBM WebSphere MQ classes for Java puede establecer el campo sslCipherSuite en la clase MQEnvironment o establecer la propiedad de entorno CMQC.SSL_CIPHER_SUITE_PROPERTY.

Configuración de la aplicación para utilizar correlaciones de IBM Java u Oracle Java CipherSuite

Desde IBM WebSphere MQ Version 7.5.0, Fixpack 5, puede configurar si la aplicación utiliza las correlaciones predeterminadas de IBM Java CipherSuite a WebSphere MQ CipherSpec , o las correlaciones de Oracle CipherSuite a WebSphere MQ CipherSpec . Por lo tanto, puede utilizar TLS CipherSuites si la aplicación utiliza un JRE de IBM o un JRE de Oracle . La propiedad del sistema Java `com.ibm.mq.cfg.useIBMCipherMappings` controla qué correlaciones se utilizan. La propiedad puede tener uno de los valores siguientes:

true
Utilice las correlaciones de IBM Java CipherSuite con WebSphere MQ CipherSpec .

Este es el valor predeterminado.

falso
Utilice las correlaciones de Oracle CipherSuite a WebSphere MQ CipherSpec .

La tabla siguiente lista las especificaciones de cifrado soportadas por IBM WebSphere MQ y sus suites de cifrado equivalentes. La tabla también indica si una aplicación IBM WebSphere MQ classes for Java puede establecer una conexión con un gestor de colas si se especifica una CipherSpec en el extremo del servidor del canal MQI y se especifica la CipherSuite equivalente en el extremo del cliente.

Tabla 89. CipherSpecs soportadas por WebSphere MQ y sus CipherSuites equivalentes

CipherSpec	**CipherSuite equivalente**	**¿Es posible la conexión si SFIPS [1] está establecido en YES?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	No
NULL_SHA	SSL_RSA_WITH_NULL_SHA	No
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5 (IBM JRE) No es equivalente para Oracle JRE.	No
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	No
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA (IBM JRE) No es equivalente para Oracle JRE.	No
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (IBM JRE) SSL_RSA_EXPORT_WITH_RC4_40_MD5 (Oracle JRE)	No
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA (IBM JRE) No es equivalente para Oracle JRE.	No
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA (IBM JRE) No es equivalente para Oracle JRE.	No
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (IBM JRE) No es equivalente para Oracle JRE.	No
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA (IBM JRE) No es equivalente para Oracle JRE.	No
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256 (IBM JRE) TLS_RSA_WITH_NULL_SHA256 (Oracle JRE)	No [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA (Oracle JRE)	Sí [5] [7]
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA256 (Oracle JRE)	Sí [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA (Oracle JRE)	Sí [5] [7]

Tabla 89. CipherSpecs soportadas por WebSphere MQ y sus CipherSuites equivalentes (continuación)		
CipherSpec	**CipherSuite equivalente**	**¿Es posible la conexión si SFIPS [1] está establecido en YES?**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA256 (Oracle JRE)	Sí [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8]	SSL_RSA_WITH_DES_CBC_SHA	No[3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8] [9]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Sí
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA (IBM JRE) No es equivalente para Oracle JRE.	No[4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (IBM JRE) No es equivalente para Oracle JRE.	No[6]

Notas:

1. En una aplicación IBM WebSphere MQ classes for Java , indique que solo se van a utilizar algoritmos certificados por FIPS estableciendo el campo sslFipsObligatorio en la clase MQEnvironment en `true` e indique que también se pueden utilizar algoritmos no certificados por FIPS estableciendo el campo sslFipsObligatorio en `false`. De forma alternativa, establezca la propiedad de entorno CMQC.SSL_FIPS_REQUIRED_PROPERTY.
2. Esta CipherSpec no tiene ninguna CipherSuiteequivalente.
3. Esta CipherSpec estaba certificada con FIPS 140-2 antes del 19th de mayo de 2007.
4. Esta CipherSpec estaba certificada con FIPS 140-2 antes del 19th de mayo de 2007. El nombre FIPS_WITH_DES_CBC_SHA es histórico y refleja el hecho de que esta CipherSpec anteriormente era compatible con FIPS (pero ya no lo es). Esta CipherSpec está en desuso y su uso no se recomienda.
5. Estas CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) no se pueden utilizar para proteger una conexión desde WebSphere MQ Explorer a un gestor de colas a menos que se apliquen los archivos de políticas no restringidas adecuados al JRE utilizado por el explorador.

   Consulte Información de seguridad para obtener más información sobre los archivos de políticas.
6. El nombre FIPS_WITH_3DES_EDE_CBC_SHA es histórico y refleja el hecho de que esta CipherSpec anteriormente era compatible con FIPS (pero ya no lo es). Esta CipherSpec está en desuso y su uso no se recomienda.
7. Estos CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) requieren IBM JRE 6.0 SR13 FP2 , 7.0 SR4 FP2 o posterior.
8. Estas CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) pueden utilizar SSLv3 o TLS. De forma predeterminada, cuando FIPS no está habilitado, se utiliza SSLv3 . Para utilizar TLS, establezca la propiedad del sistema Java **com.ibm.mq.cfg.preferTLS** en `true`.

9. 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.

Información relacionada

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Federal Information Processing Standards (FIPS) para UNIX, Linux y Windows

Blog de MQdev: MQ Java, TLS Ciphers, Non-IBM JRE & APARs IT06775, IV66840, IT09423, IT10837

Blog de MQdev: La relación entre MQ CipherSpecs y Java Cipher Suites

Ejecución de clases de WebSphere MQ para aplicaciones Java

Si escribe una aplicación (una clase que contiene un método main ()), utilizando el cliente o la modalidad de enlaces, ejecute el programa utilizando el intérprete Java.

Utilice el mandato:

```
java -Djava.library.path=library_path MyClass
```

donde *vía_acceso_biblioteca* es la vía de acceso a las clases de WebSphere MQ para bibliotecas Java (consulte Las clases de WebSphere MQ para bibliotecas Java).

WebSphere MQ classes for Java comportamiento dependiente del entorno

Las clases WebSphere MQ para Java le permiten crear aplicaciones que se pueden ejecutar en distintas versiones de WebSphere MQ. Esta colección de temas describe el comportamiento de las clases Java que dependen de estas diferentes versiones.

WebSphere MQ classes for Java proporciona un núcleo de clases, 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.

Excepto donde se indique aquí, el comportamiento mostrado es el que se describe en la publicación Application Programming Reference correspondiente al gestor de colas.

Clases principales en WebSphere MQ classes for Java

WebSphere MQ classes for Java contiene un conjunto principal de clases, que se puede 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 WebSphere MQ classes for Java” en la página 724.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Excepto:

  – MatchOptions
  – GroupStatus
  – SegmentStatus
  – Segmentation
- 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 WebSphere MQ classes for Java" en la página 724 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. (Consulte "Opciones de conexión para clases de WebSphere MQ para Java" en la página 667.)

Restricciones y variaciones para clases principales de WebSphere 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 Windows, UNIX o Linux WebSphere MQ , excepto para las siguientes restricciones y variaciones menores.

*Restricciones para valores MQGMO_ * en clases WebSphere MQ para 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 está soportado cuando se utiliza desde Java.

*Restricciones para valores MQPMRF_ * en clases WebSphere MQ para 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 z/OS no dan soporte a listas de distribución.

*Restricciones para valores MQPMO_ * en clases WebSphere MQ para 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 valores MQCNO_ * en clases WebSphere MQ para Java*
Determinados valores `MQCNO_*` no están soportados.

- La reconexión automática de cliente no está soportada por las clases de WebSphere MQ para 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 valores MQRO_ * en clases WebSphere MQ para 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.

Características fuera de las clases principales de WebSphere MQ classes for Java

WebSphere MQ classes for Java contiene determinadas funciones diseñadas específicamente para utilizar extensiones de API que no están soportadas por todos los gestores de colas. Esta colección de temas describe cómo se comportan al utilizar un gestor de colas que *no* les da soporte.

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 WebSphere MQ en sistemas UNIX, Linuxo Windows en modalidad de enlaces. En otro caso, el método falla y devuelve el error MQRC_ENVIRONMENT_ERROR.

Consulte "Coordinación JTA/JDBC utilizando clases WebSphere MQ para Java" en la página 712 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
Segmentation

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 clases de WebSphere MQ para 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 concreto, los gestores de colas de z/OS no dan soporte a 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 campos MQMD con clases WebSphere MQ para 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 ejecute en un gestor de colas que no dé soporte a la agrupación y segmentación de mensajes.

Si una aplicación Java intenta obtener () un mensaje de un gestor de colas que no soporta estos campos, y el mensaje físico que se va a recuperar forma parte de un grupo de mensajes segmentados (es decir, tiene valores no predeterminados para los campos MQMD), se recupera sin errores. 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 las clases de WebSphere MQ para Java en CICS Transaction Server
En el entorno CICS Transaction Server para z/OS , sólo la hebra principal (primera) puede emitir llamadas CICS o WebSphere MQ .

Tenga en cuenta que las clases JMS de WebSphere MQ no están soportadas para su uso en una aplicación Java CICS.

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.

Ejecución de clases IBM WebSphere MQ para aplicaciones Java en la plataforma Java Enterprise Edition
Existen determinadas restricciones y consideraciones de diseño que deben tenerse en cuenta antes de utilizar las clases de IBM WebSphere MQ para Java en Java EE

Las clases IBM WebSphere MQ para Java tienen restricciones cuando se utilizan en un entorno Java EE . También hay consideraciones adicionales que deben tenerse en cuenta al diseñar, implementar y

gestionar una aplicación IBM WebSphere 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 clases IBM WebSphere MQ para Java es el propio IBM WebSphere MQ . Aunque una aplicación bajo el control de JTA puede utilizar clases IBM WebSphere MQ para Java, cualquier trabajo realizado a través de estas clases no está controlado por las unidades de trabajo 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 restricción se aplica a las transacciones gestionadas por aplicación o bean y a las transacciones gestionadas por contenedor y a todos los contenedores Java EE . Para realizar el trabajo de mensajería directamente con IBM WebSphere MQ dentro de las transacciones coordinadas por el servidor de aplicaciones, en su lugar se deben utilizar las clases IBM WebSphere MQ para JMS.

Creación de hebras

IBM WebSphere MQ classes for Java crea hebras internamente para varias operaciones. Por ejemplo, cuando se ejecuta en modalidad BINDINGS para llamar directamente a un gestor de colas local, las llamadas se realizan en una hebra 'worker' creada internamente por las clases IBM WebSphere MQ para 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 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 clases IBM WebSphere MQ para Java, es posible que el servidor de aplicaciones no pueda distinguir entre el código de aplicación y las clases IBM WebSphere MQ para el código Java , por lo que las hebras descritas anteriormente hacen que la aplicación no sea compatible con la especificación de contenedor. Las clases IBM WebSphere MQ para JMS no interrumpen estas especificaciones Java EE y, por lo tanto, se pueden utilizar en su lugar.

Limitaciones de seguridad

Las políticas de seguridad implementadas por un servidor de aplicaciones pueden impedir determinadas operaciones realizadas por las clases IBM WebSphere MQ para la API Java , como por ejemplo crear y operar 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, las clases IBM WebSphere MQ para Java pueden romper las reglas de hebras de política de seguridad de Java definidas para el servidor de aplicaciones, y es posible que la API no pueda crear todas las hebras que necesita para funcionar. Para evitar problemas con la gestión de hebras, el uso de clases de IBM WebSphere MQ para Java no está soportado en entornos en los que la seguridad de Java está habilitada.

Consideraciones sobre el aislamiento de las aplicaciones

Una ventaja prevista de ejecutar aplicaciones dentro de un entorno Java EE es el aislamiento de aplicaciones. El diseño y la implementación de clases IBM WebSphere MQ para Java anteriores al entorno Java EE . IBM WebSphere MQ las clases para Java se pueden utilizar de una forma que no dé 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
  - Configuración SSL 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 ejecuta en un entorno de varias aplicaciones 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 utilizar de forma predeterminada las propiedades configuradas en la clase MQEnvironment de todo el proceso.
- La clase MQEnvironment introduce una serie de métodos estáticos que actúan globalmente en todas las aplicaciones que utilizan clases IBM WebSphere MQ para Java dentro del mismo proceso JVM, y no hay forma de alterar temporalmente este comportamiento para aplicaciones concretas. Estos son algunos ejemplos:
  - configuración de propiedades SSL, 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 estos métodos afecta a todas las aplicaciones que se ejecutan en el mismo entorno 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, como la sustitución del gestor de conexiones predeterminado para una aplicación utilizando el método MQEnvironment.setDefaultConnectionManager(), por lo tanto, afectan a otras aplicaciones que se ejecutan en el mismo servidor de aplicaciones Java EE .
- SSL se configura para aplicaciones que utilizan clases IBM WebSphere MQ para Java utilizando la clase MQEnvironment y las propiedades de objeto MQQueueManager . No está integrado con la configuración de seguridad gestionada del propio servidor de aplicaciones. Debe asegurarse de que configura las clases IBM WebSphere MQ para Java adecuadamente para proporcionar el nivel de seguridad necesario y no utilizar la configuración del servidor de aplicaciones.

Restricciones de la modalidad de enlaces

IBM WebSphere MQ y WebSphere Application Server se pueden instalar en la misma máquina de forma que las versiones principales del gestor de colas y del adaptador de recursos (RA) de IBM WebSphere MQ que se suministran en WebSphere Application Server son diferentes. Por ejemplo, WebSphere Application Server Version 7.0, que suministra un nivel RA de IBM WebSphere MQ de 7.0.1, se puede instalar en la misma máquina que un gestor de colas de Version 6.0 .

Si las versiones principales del gestor de colas y del adaptador de recursos son diferentes, no se pueden utilizar conexiones de enlaces. Cualquier conexión de WebSphere Application Server con el gestor de colas utilizando el adaptador de recursos debe utilizar conexiones de tipo de cliente. Se pueden utilizar conexiones de enlaces si las versiones son iguales.

Utilización de clases de WebSphere MQ para JMS

WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) es el proveedor JMS que se proporciona con WebSphere MQ. Además de implementar las interfaces definidas en el paquete javax.jms , WebSphere MQ classes for JMS proporciona dos conjuntos de extensiones para la API JMS.

La especificación JMS define un conjunto de interfaces que las aplicaciones pueden utilizar para realizar operaciones de mensajería. El paquete javax.jms define las interfaces JMS y un proveedor JMS implementa estas interfaces para un producto de mensajería específico. WebSphere MQ versión 7.5

utiliza actualmente la especificación JMS 1.1 . WebSphere MQ classes for JMS es un proveedor JMS que implementa las interfaces JMS para WebSphere MQ.

La especificación JMS espera que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene objetos administrados en un repositorio central, y una aplicación JMS recupera estos objetos utilizando JNDI (Java Naming and Directory Interface). WebSphere MQ classes for JMS da soporte al uso de objetos administrados, y un administrador puede utilizar la herramienta de administración JMS de WebSphere MQ o WebSphere MQ Explorer para crear y mantener objetos administrados.

WebSphere 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 JMS de WebSphere MQ
Los releases anteriores de WebSphere MQ classes for JMS contienen extensiones que se implementan en objetos como MQConnectionFactory, MQQueue y MQTopic. Estos objetos tienen propiedades y métodos específicos de WebSphere MQ. Los objetos pueden ser objetos administrados, o una aplicación puede crearlos dinámicamente en tiempo de ejecución. Este release de WebSphere MQ classes for JMS mantiene estas extensiones, que ahora se conocen como extensiones JMS de WebSphere MQ . Puede continuar utilizando, sin alteraciones, aplicaciones que hacen uso de estas extensiones.

Las extensiones JMS de IBM
Este release de WebSphere MQ classes for JMS proporciona un conjunto más genérico de extensiones para la API JMS, que no son específicas de WebSphere MQ como sistema de mensajería. Estas extensiones se conocen como las extensiones JMS de IBM y tienen los siguientes objetivos generales:

- Para proporcionar un mayor nivel de coherencia entre los proveedores JMS de IBM
- Para facilitar la escritura de una aplicación puente entre dos sistemas de mensajería IBM
- Para facilitar el puerto de una aplicación desde un proveedor JMS de IBM a otro

Las extensiones proporcionan una función similar a la proporcionada en Message Service Client for C/C++ y Message Service Client for .NET.

¿Por qué debo utilizar WebSphere MQ classes for JMS?

El uso de WebSphere MQ classes for JMS tiene las ventajas siguientes:

- Puede reutilizar las habilidades JMS.

  WebSphere MQ classes for JMS es un proveedor JMS que implementa las interfaces JMS para WebSphere MQ como sistema de mensajería. Si su organización es nueva en WebSphere MQ, pero ya tiene conocimientos de desarrollo de aplicaciones JMS, es posible que le resulte más fácil utilizar la API JMS familiar para acceder a los recursos de WebSphere MQ en lugar de una de las otras API proporcionadas con WebSphere MQ.
- JMS es una parte integral de Java Platform, Enterprise Edition (Java EE).

  JMS es la API natural que se utiliza para la mensajería en la plataforma Java EE . Cada servidor de aplicaciones que sea compatible con Java EE debe incluir un proveedor JMS. Puede utilizar JMS en clientes de aplicaciones, servlets, páginas JavaServer (JSP), enterprise Java beans (EJB) y beans controlados por mensajes (MDB). Tenga en cuenta en particular que las aplicaciones Java EE utilizan MDB para procesar mensajes de forma asíncrona, y todos los mensajes se entregan a MDB como mensajes JMS.
- Un administrador puede crear y mantener objetos administrados JMS en un repositorio central, y las clases WebSphere MQ para aplicaciones JMS pueden recuperar estos objetos utilizando JNDI (Java Naming and Directory Interface).

  Las fábricas de conexiones JMS y los destinos encapsulan información específica de WebSphere MQ como, 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 los destinos se almacenan como objetos administrados, esta información no se codifica en una aplicación. Por lo tanto, esta disposición proporciona a la aplicación un grado de independencia de la configuración subyacente de WebSphere MQ .

- JMS es una API estándar del sector 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. A continuación, la aplicación es totalmente independiente de cualquier proveedor JMS, como por ejemplo WebSphere MQ classes for JMS, y se puede portar de un proveedor JMS a otro sin ningún cambio en la aplicación.

  Si JNDI no está disponible en un entorno de aplicación determinado, una aplicación WebSphere MQ classes for JMS puede utilizar extensiones para la API JMS para crear y configurar fábricas de conexiones y destinos dinámicamente en tiempo de ejecución. A continuación, la aplicación es completamente autocontenida, pero está vinculada a WebSphere MQ classes for JMS como proveedor JMS.

- Las aplicaciones de puente pueden ser más fáciles de escribir utilizando JMS.

  Una aplicación puente es una aplicación que recibe mensajes de un sistema de mensajería y los envía a otro sistema de mensajería. La escritura de una aplicación puente puede ser complicada utilizando API y formatos de mensajes específicos del producto. En su lugar, puede escribir una aplicación puente utilizando dos proveedores JMS, uno para cada sistema de mensajería. A continuación, la aplicación sólo utiliza una API, la API JMS, y sólo procesa mensajes JMS.

Iniciación a WebSphere MQ classes for JMS

Este tema proporciona una visión general de las clases de WebSphere MQ para JMS e indica lo que necesita saber antes de utilizar las clases de WebSphere MQ para JMS.

Requisitos previos para WebSphere MQ classes for JMS

Para desarrollar y ejecutar WebSphere MQ clases para aplicaciones JMS, necesita determinados componentes de software como requisitos previos.

Para obtener la información más reciente sobre los requisitos previos para WebSphere MQ classes for JMS, consulte el archivo léame de WebSphere MQ .

Para desarrollar WebSphere MQ clases para aplicaciones JMS, necesita un kit de desarrollo de software (SDK) de Java 2. Los detalles de los JDK soportados con el sistema operativo se pueden encontrar en la página de requisitos del sistema WebSphere MQ . Consulte Requisitos de WebSphere MQ.

Para ejecutar las clases WebSphere MQ para aplicaciones JMS, necesita los siguientes componentes de software:

- Un gestor de colas de WebSphere MQ
- Un Java Runtime Environment (JRE), para cada sistema en el que ejecuta aplicaciones

Si necesita que las conexiones SSL utilicen módulos criptográficos con la certificación FIPS 140-2, necesita el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS). Cada IBM Java 2 SDK y JRE de la versión 5 o posterior contiene IBMJSSEFIPS.

Puede utilizar las direcciones de Internet Protocol Versión 6 (IPv6) en las clases WebSphere MQ para aplicaciones JMS proporcionadas por las direcciones IPv6 están soportadas por la máquina virtual Java (JVM) y la implementación TCP/IP en el sistema operativo. La herramienta de administración JMS de WebSphere MQ (consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953) también acepta direcciones IPv6 .

La herramienta de administración JMS de WebSphere MQ y WebSphere MQ Explorer utilizan JNDI (Java Naming and Directory Interface) para acceder a un servicio de directorio, que almacena objetos administrados. Las clases de WebSphere MQ para aplicaciones JMS también pueden utilizar JNDI para recuperar objetos administrados de un servicio de directorio. Un proveedor de servicios es un código que

proporciona acceso a un servicio de directorio correlacionando llamadas JNDI con llamadas al servicio de directorio. Los proveedores de servicios siguientes se proporcionan con clases WebSphere MQ para JMS:

- Un proveedor de servicios LDAP (Lightweight Directory Access Protocol) en los archivos ldap.jar y providerutil.jar. El proveedor de servicios LDAP proporciona acceso a un servicio de directorio basado en un servidor LDAP.
- Un proveedor de servicios del sistema de archivos en los archivos fscontext.jar y providerutil.jar. El proveedor de servicios del sistema de archivos da 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 concreto, debe configurar el servidor LDAP para almacenar objetos Java. Si desea obtener información sobre cómo instalar y configurar el servidor LDAP, consulte la documentación que se suministra con el servidor.

Preparación de programas JMS para el cliente IBM WebSphere MQ para HP Integrity NonStop Server

En este tema se explica lo que necesita saber antes de desarrollar y ejecutar programas JMS para el cliente IBM WebSphere MQ para HP Integrity NonStop Server.

Las clases IBM WebSphere MQ para JMS se instalan como parte de la instalación del cliente IBM WebSphere MQ para HP Integrity NonStop Server . Para obtener un resumen del contenido de la instalación, consulte Sistema de archivos.

Algunos aspectos de la funcionalidad del cliente son específicos del sistema operativo del sistema principal. Para obtener más información sobre las características soportadas para el cliente de IBM WebSphere MQ para HP Integrity NonStop Server, consulte Cliente deIBM WebSphere MQ para entornos y características soportados de HP Integrity NonStop Server.

Requisitos previos

Para crear y ejecutar aplicaciones JMS, el componente *HP Integrity NonStop Server for Java* debe estar instalado y disponible.

Configuración

Para obtener información sobre cómo configurar el entorno para ejecutar y crear aplicaciones en las que puede utilizar las clases IBM WebSphere MQ para JMS, consulte “Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS” en la página 737.

Para obtener información sobre los pasos necesarios para configurar un gestor de colas para que acepte conexiones de las aplicaciones cliente, consulte “Configuración posterior a la instalación para las clases de WebSphere MQ para aplicaciones JMS” en la página 788.

Para obtener información sobre la validación de las clases IBM WebSphere MQ para el entorno JMS, consulte “La prueba de verificación de instalación punto a punto para las clases WebSphere MQ para JMS” en la página 791.

Escritura de aplicaciones

Para obtener más información sobre cómo escribir aplicaciones JMS, consulte “Escritura de clases de WebSphere MQ para aplicaciones JMS” en la página 822.

Para obtener más información sobre cómo utilizar la herramienta de administración JMS de IBM WebSphere MQ , consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953.

Ejemplos

Se proporcionan aplicaciones de ejemplo en el siguiente subdirectorio de la instalación: `opt/mqm/samp/jms`.

Para obtener más información sobre los pasos de configuración necesarios para ejecutar los ejemplos, consulte “Preparación y ejecución de los programas de ejemplo” en la página 112.

Resolución de problemas

Para obtener información sobre cómo resolver problemas, consulte “Resolución de problemas con clases IBM WebSphere MQ para JMS” en la página 813.

Instalación y configuración de WebSphere MQ classes for JMS

En esta sección se describen los directorios y archivos que se crean al instalar WebSphere MQ classes for JMS e indica cómo configurar WebSphere MQ classes for JMS después de la instalación.

Conceptos relacionados

“Qué se instala para las clases IBM WebSphere MQ para JMS” en la página 734
Se crean varios archivos y directorios al instalar las clases de IBM WebSphere MQ para JMS. En Windows , se realiza alguna configuración durante la instalación estableciendo automáticamente variables de entorno. En otras plataformas, y en determinados entornos Windows, debe establecer las variables de entorno antes de poder ejecutar las clases de IBM WebSphere MQ para aplicaciones JMS.

“Ejecución de clases de WebSphere MQ para aplicaciones JMS bajo el gestor de seguridad de Java” en la página 744
Las clases de WebSphere MQ para JMS se pueden ejecutar con el gestor de seguridad de Java habilitado. Para ejecutar aplicaciones correctamente con el gestor de seguridad habilitado, debe configurar la máquina virtual Java (JVM) con un archivo de configuración de políticas adecuado.

“El adaptador de recursos de IBM WebSphere MQ” en la página 748
El adaptador de recursos permite que las aplicaciones que se ejecutan en un servidor de aplicaciones accedan a los recursos de IBM WebSphere MQ . El adaptador de recursos da soporte a la comunicación de entrada y de salida.

“Configuración posterior a la instalación para las clases de WebSphere MQ para aplicaciones JMS” en la página 788
Este tema le indica qué autorizaciones necesitan las clases WebSphere MQ para las aplicaciones 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.

“La prueba de verificación de instalación punto a punto para las clases WebSphere MQ para JMS” en la página 791
Se proporciona un programa de prueba de verificación de instalación punto a punto (IVT) con clases WebSphere MQ para JMS. 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.

“La prueba de verificación de la instalación de publicación/suscripción para las clases WebSphere MQ para JMS” en la página 795
Se proporciona un programa de prueba de verificación de instalación (IVT) de publicación/suscripción con clases WebSphere MQ para 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.

“El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ” en la página 799

El programa IVT se proporciona como un archivo EAR. Para utilizar el programa, debe desplegarlo y definir algunos objetos como recursos JCA.

"Configuración del adaptador de recursos para la comunicación de salida" en la página 767
Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

"Soporte para OSGi" en la página 812
OSGi proporciona una infraestructura que da soporte al despliegue de aplicaciones como paquetes. Se proporcionan nueve paquetes OSGi como parte de IBM WebSphere MQ classes for JMS.

"Resolución de problemas con clases IBM WebSphere MQ para JMS" en la página 813
Puede investigar problemas ejecutando los programas de verificación de la instalación y utilizando los recursos de rastreo y de registro.

Tareas relacionadas

"Instalación y prueba del adaptador de recursos MQ en WAS CE" en la página 802
Instalación del adaptador de recursos de IBM WebSphere MQ y ejecución de la aplicación de prueba de verificación de instalación (IVT) en WebSphere Application Server CE.

"Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado" en la página 804
Si desea utilizar una cola, un gestor de colas, un puerto, un host, un canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, debe modificar la aplicación IVT y los scripts asociados en WebSphere Application Server CE antes de desplegar el adaptador de recursos o la aplicación IVT.

"Despliegue de la aplicación IVT en JBoss con un entorno IBM WebSphere MQ personalizado" en la página 807
Al instalar el adaptador de recursos de IBM WebSphere MQ en JBoss, si desea utilizar una cola, gestor de colas, puerto, host, canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, primero debe modificar la aplicación IVT y los scripts asociados en JBoss antes de desplegar el adaptador de recursos o la aplicación IVT.

Determinación de problemas para el adaptador de recursos IBM WebSphere MQ

Referencia relacionada

"Scripts proporcionados con WebSphere MQ classes for JMS" en la página 811
Se proporcionan varios scripts para ayudar con las tareas comunes que se deben realizar cuando se utilizan clases WebSphere MQ para JMS.

Qué se instala para las clases IBM WebSphere MQ para JMS

Se crean varios archivos y directorios al instalar las clases de IBM WebSphere MQ para JMS. En Windows , se realiza alguna configuración durante la instalación estableciendo automáticamente variables de entorno. En otras plataformas, y en determinados entornos Windows, debe establecer las variables de entorno antes de poder ejecutar las clases de IBM WebSphere MQ para aplicaciones JMS.

Para la mayoría de los sistemas operativos, las clases IBM WebSphere MQ para JMS se instalan como un componente opcional al instalar IBM WebSphere MQ. Para el cliente de IBM WebSphere MQ para HP Integrity NonStop Server, las clases IBM WebSphere MQ para JMS se instalan de forma predeterminada. Para obtener más información sobre la instalación de IBM WebSphere MQ, consulte:

Instalación de un servidor WebSphere MQ
Instalación de un cliente IBM WebSphere MQ

La Tabla 90 en la página 734 muestra dónde se instalan las clases IBM WebSphere MQ para los archivos JMS en cada plataforma.

Tabla 90. Directorios de instalación de IBM WebSphere MQ para JMS

Plataforma	**Directorio**
AIX	*MQ_INSTALLATION_PATH*/java
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/java

Tabla 90. Directorios de instalación de IBM WebSphere MQ para JMS (continuación)	
Plataforma	**Directorio**
HP-UX, Linuxy Solaris	*MQ_INSTALLATION_PATH*/java
Windows	*MQ_INSTALLATION_PATH*\java
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.	

El directorio de instalación incluye:

- Las clases IBM WebSphere MQ para archivos JAR JMS, que se encuentran en el directorio *MQ_INSTALLATION_PATH*\java\lib.
- Las bibliotecas nativas de IBM WebSphere MQ , que utilizan las aplicaciones que utilizan 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 WebSphere MQ, consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 739.
- Scripts adicionales descritos en “Scripts proporcionados con WebSphere MQ classes for JMS” en la página 811. Estos scripts se encuentran en el directorio *MQ_INSTALLATION_PATH*\java\bin.
- Las especificaciones de las clases IBM WebSphere MQ para la API JMS. La herramienta Javadoc se ha utilizado para generar las páginas HTML que contienen las especificaciones de la API.

  Las páginas HTML se encuentran en el directorio *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses.

  En sistemas UNIX, Linuxy Windows, este subdirectorio contiene las páginas HTML individuales.
- Soporte de OGSi. Los paquetes OSGi se instalan en el directorio java \lib\OSGi y se describen en “Soporte para OSGi” en la página 812.
- El adaptador de recursos de IBM WebSphere MQ , que se puede desplegar en cualquier servidor de aplicaciones compatible con JCA 1.5 (o posterior).

  El adaptador de recursos de IBM WebSphere MQ se encuentra en el directorio *MQ_INSTALLATION_PATH*\java\lib\jca; para obtener más información, consulte “El adaptador de recursos de IBM WebSphere MQ” en la página 748
- En Windows, los símbolos que se pueden utilizar para la depuración se instalan en el directorio *MQ_INSTALLATION_PATH*\java\lib\símbolos.

El directorio de instalación también incluye algunos archivos que pertenecen a otros componentes de IBM WebSphere MQ. Estos directorios son los siguientes:

- El transporte IBM WebSphere MQ para SOAP, que proporciona un transporte JMS para SOAP, se instala en el directorio *MQ_INSTALLATION_PATH*\java\lib\soap. Para obtener más información sobre el transporte IBM WebSphere MQ para SOAP, consulte la sección del centro de información que describe “Transporte de WebSphere MQ para SOAP” en la página 966.
- En plataformas distribuidas, IBM WebSphere MQ Bridge for HTTP se instala en el directorio *MQ_INSTALLATION_PATH*\java\lib\http. Para obtener más información sobre el puente IBM WebSphere MQ para HTTP, consulte la sección del centro de información que describe “Puente WebSphere MQ para HTTP” en la página 1042

Algunas aplicaciones de ejemplo se proporcionan con clases IBM WebSphere MQ para JMS. La Tabla 91 en la página 736 muestra dónde se instalan las aplicaciones de ejemplo en cada plataforma.

Tabla 91. Directorios de ejemplos	
Plataforma	**Directorio**
AIX	*MQ_INSTALLATION_PATH*/samp/jms
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/samp/jms
HP-UX, Linuxy Solaris	*MQ_INSTALLATION_PATH*/samp/jms
Windows	*MQ_INSTALLATION_PATH*\tools\jms
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.	

Después de la instalación, es posible que deba realizar algunas tareas de configuración para compilar y ejecutar aplicaciones.

"Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS" en la página 737 describe la vía de acceso de clases necesaria para ejecutar clases simples de IBM WebSphere MQ para aplicaciones JMS. 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 debe establecer para ejecutar los scripts proporcionados con las clases IBM WebSphere MQ para JMS.

Si necesita que la aplicación IBM WebSphere MQ classes for JMS enlace con código escrito en lenguajes distintos de Java (por ejemplo, para utilizar el transporte de enlaces al conectarse a un gestor de colas), "Configuración de las bibliotecas JNI (Java Native Interface)" en la página 739 explica dónde encontrar la ubicación de las bibliotecas JNI (Java Native Interface) para especificar como parámetro del mandato Java.

Para controlar propiedades como, por ejemplo, el rastreo y registro de una aplicación, debe proporcionar un archivo de propiedades de configuración. Las clases IBM WebSphere MQ para el archivo de propiedades de configuración JMS se describen en "El archivo de configuración IBM WebSphere MQ classes for JMS" en la página 741.

Instalación y actualización de las clases WebSphere MQ para archivos JAR JMS

La única forma soportada de obtener las clases IBM WebSphere MQ para archivos JAR JMS en un sistema es instalar el producto IBM WebSphere MQ o WebSphere MQ V7.5 Clients SupportPac-MQC75, o utilizando una herramienta de gestión de software como Apache Maven, para obtener más información consulte "IBM WebSphere MQ classes for JMS y herramientas de gestión de software" en la página 743.

No mueva, ni copie, las clases IBM WebSphere MQ para archivos JAR JMS o bibliotecas nativas, a otras máquinas o a una ubicación diferente en una máquina donde se hayan instalado las clases IBM WebSphere MQ para JMS, a menos que esté utilizando una herramienta de gestión de software.

- Los fixpacks no se pueden aplicar a una "instalación" en la que los archivos JAR se han copiado desde otra máquina, porque esto hace que sea mucho más difícil asegurarse de que todos los archivos JAR se mantienen en el paso uno con el otro y están en niveles compatibles.
- La copia de las clases IBM WebSphere MQ para archivos JAR JMS entre máquinas también puede dar como resultado varias copias de los archivos que residen en la misma máquina, lo que puede provocar problemas al dar servicio al código y depurar problemas.
- El mandato dspmqver , utilizado para visualizar información de versión de una instalación de IBM WebSphere MQ , sólo muestra información de versión para las clases IBM WebSphere MQ para JMS instaladas en el directorio `\java\lib` .

  Si varias copias de los archivos residen en la misma máquina, es posible que la ejecución de **dspmqver** no proporcione información precisa sobre la versión de las clases IBM WebSphere MQ para JMS que utiliza una aplicación.

No incluya las clases IBM WebSphere MQ para los archivos JAR JMS dentro de los archivadores de aplicación (por ejemplo, archivadores de aplicación empresarial o archivos EAR).

- Las actualizaciones de las clases IBM WebSphere MQ para JMS no se pueden aplicar utilizando un fixpack de IBM WebSphere MQ .
- El soporte de IBM no puede determinar fácilmente la versión de las clases IBM WebSphere MQ para JMS que utiliza la aplicación.
- Pueden surgir problemas si varias aplicaciones que se ejecutan en el mismo Java Runtime Environment incluyen distintas versiones de las clases IBM WebSphere MQ para JMS, ya que varias versiones de las clases IBM WebSphere MQ para JMS se cargan en el Java Runtime Environment al mismo tiempo.

  Entre los ejemplos de estos problemas se incluyen las siguientes excepciones:

```
java.lang.ClassCastException :
com.ibm.mq.jmqi.system.JmqiSystemEnvironment incompatible with
com.ibm.mq.jmqi.system.JmqiSystemEnvironment

java.lang.ClassCastException :
com.ibm.mq.jms.MQQueue incompatible with com.ibm.mq.jms.MQQueue
```

- Si una aplicación utiliza el transporte BINDINGS para conectarse a un gestor de colas, cualquier actualización importante al gestor de colas también requiere que la aplicación se actualice para incluir el nivel correspondiente de las clases IBM WebSphere MQ para JMS.

  Por ejemplo, si un gestor de colas se actualiza al nivel IBM WebSphere MQ Versión 7.5 , las aplicaciones que se conectan al gestor de colas utilizando el transporte BINDINGS también deben actualizarse para incluir las clases IBM WebSphere MQ Versión 7.5 para JMS.

Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS

Para poder compilar y ejecutar clases IBM WebSphere MQ para aplicaciones JMS, el valor de la variable de entorno CLASSPATH debe incluir las clases IBM WebSphere MQ para el archivo JAR (Java Archive) JMS. Dependiendo de sus necesidades, puede ser necesario añadir otros archivos JAR a CLASSPATH. Para ejecutar los scripts proporcionados con las clases IBM WebSphere MQ para JMS, se deben establecer otras variables de entorno.

Para compilar y ejecutar clases IBM WebSphere MQ para aplicaciones JMS, utilice el valor CLASSPATH para la plataforma tal como se muestra en la Tabla 92 en la página 737. El valor incluye el directorio de ejemplos, para que pueda compilar y ejecutar las clases IBM WebSphere MQ para aplicaciones de ejemplo JMS. Como alternativa, puede especificar la vía de acceso de clases en el mandato **java** en lugar de utilizar la variable de entorno.

Tabla 92. Valor CLASSPATH para compilar y ejecutar clases IBM WebSphere MQ para aplicaciones JMS, incluidas las aplicaciones de ejemplo

Plataforma	**Valor de CLASSPATH**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP Integrity NonStop Server	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP-UX, Linuxy Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms;

Tabla 92. Valor CLASSPATH para compilar y ejecutar clases IBM WebSphere MQ para aplicaciones JMS, incluidas las aplicaciones de ejemplo (continuación)	
Plataforma	**Valor de CLASSPATH**
z/OS	CLASSPATH=*MQ_INSTALLATION_PATH*/mqm/V7R0M0/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V6R0M0/java/samples/jms:
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.	

El manifiesto del archivo JAR com.ibm.mqjms.jar contiene referencias a la mayoría de los otros archivos JAR que necesitan las clases IBM WebSphere MQ para las aplicaciones JMS y, por lo tanto, no es necesario que añada estos archivos JAR a la vía de acceso de clases. Estos archivos JAR incluyen los que necesitan las aplicaciones que utilizan JNDI (Java Naming and Directory Interface) para recuperar objetos administrados de un servicio de directorio y las aplicaciones que utilizan JTA (Java Transaction API).

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 las clases IBM WebSphere MQ para el archivo JAR Java, com.ibm.mq.jar, a la vía de acceso de clases.
- Si compila el código Java utilizando un kit de desarrollo de software (SDK) de Java 2 en la versión 1.4.2, debe añadir los siguientes archivos JAR a la vía de acceso de clases:
  - jms.jar
  - com.ibm.mq.jmqi.jar

  Además, si la aplicación utiliza JNDI para recuperar objetos administrados de un servicio de directorio, también debe añadir los siguientes archivos JAR a la vía de acceso de clases:
  - fscontext.jar
  - jndi.jar
  - ldap.jar
  - providerutil.jar

  Y si la aplicación utiliza JTA, también debe añadir jta.jar a la vía de acceso de clases.

  Tenga en cuenta que estos archivos JAR adicionales sólo son necesarios para compilar las aplicaciones, no para ejecutarlas.

Si compila utilizando la opción -Xlint, puede recibir un mensaje para avisarle de que com.ibm.mq.ese.jar no está presente. Puede pasar por alto este aviso. Este archivo sólo está presente si ha instalado Extended Security Edition.

Los scripts proporcionados con las clases IBM WebSphere MQ para JMS 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 están instaladas las clases WebSphere MQ para JMS.

MQ_JAVA_LIB_PATH
Esta variable de entorno especifica el directorio donde se almacenan las clases de WebSphere MQ para bibliotecas JMS, tal como se muestra en la Tabla 93 en la página 739.

En Windows, todas las variables de entorno se establecen automáticamente durante la instalación. En cualquier otra plataforma, debe establecerlas usted mismo.

Para establecer las variables de entorno si está utilizando una JVM de 32 bits en sistemas UNIX, HP Integrity NonStop Server o Linux, puede utilizar el script setjmsenv. Para establecer las variables de entorno si está utilizando una JVM de 64 bits en un sistema UNIX o Linux, puede utilizar el script setjmsenv64. Estos scripts se encuentran en el directorio *MQ_INSTALLATION_PATH*/java/bin , donde *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado IBM WebSphere MQ .

Puede utilizar el script setjmsenv o setjmsenv64 de varias formas: puede utilizarlo como base para definir las variables de entorno necesarias, 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. De forma alternativa, puede ejecutar el script en cada sesión desde la que se van a ejecutar los scripts de inicio JMS. Si elige esta opción, tendrá que ejecutar el script en cada ventana de shell que inicie, durante el proceso de verificación de JMS escribiendo `. ./setjmsenv` o `. ./setjmsenv64`

Configuración de las bibliotecas JNI (Java Native Interface)

Las clases IBM WebSphere MQ para aplicaciones 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 de canal escritos en lenguajes distintos de Java, deben ejecutarse en un entorno que acceda a las bibliotecas JNI (Java Native Interface).

Acerca de esta tarea

Para configurar este entorno, debe configurar la vía de acceso de biblioteca del entorno para que la máquina virtual Java (JVM) pueda cargar la biblioteca mqjbnd antes de iniciar las clases IBM WebSphere MQ para la aplicación JMS.

IBM WebSphere 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 clases de IBM WebSphere MQ para JMS y el gestor de colas. La biblioteca mqjbnd instalada con IBM WebSphere MQ Versión 7.5 se puede utilizar para conectarse a cualquier gestor de colas IBM WebSphere MQ Versión 7.5 (o anterior).

mqjexitstub02
La biblioteca mqjexitstub02 se carga mediante las clases IBM WebSphere MQ para JMS 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 distinto de Java.

En determinadas plataformas, IBM WebSphere MQ instala las versiones de 32 bits y 64 bits de estas bibliotecas JNI. La ubicación de las bibliotecas para cada plataforma se muestra en la Tabla 1

Tabla 93. La ubicación de las clases IBM WebSphere MQ para bibliotecas JMS para cada plataforma

Plataforma	**Directorio que contiene las clases de IBM WebSphere MQ para bibliotecas JMS**
AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
HP-UX	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)

Tabla 93. La ubicación de las clases IBM WebSphere MQ para bibliotecas JMS para cada plataforma (continuación)	
Plataforma	**Directorio que contiene las clases de IBM WebSphere MQ para bibliotecas JMS**
Linux (POTENCIA , x86-64 y zSeries s390x plataformas)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Linux (plataformax86) Linux (Plataforma zSeries)	*MQ_INSTALLATION_PATH* /java/lib
Solaris (plataformasx86-64 y SPARC)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
Windows	*MQ_INSTALLATION_PATH*\java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\java\lib64 (bibliotecas de 64 bits)
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que se instala IBM WebSphere MQ.	

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 WebSphere MQ. Por ejemplo, para una JVM de 64 bits y una ubicación de instalación de IBM WebSphere MQ predeterminada, puede utilizar los valores siguientes:

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

     Solaris HP-UX 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. Después de configurar el entorno de 32 bits o de 64 bits, inicie las clases IBM WebSphere MQ para la aplicación JMS utilizando el mandato:

   ```
   java application-name
   ```

   donde *nombre-aplicación* es el nombre de las clases IBM WebSphere MQ para la aplicación JMS que se va a ejecutar.

   Las clases de IBM WebSphere MQ para JMS lanzan una excepción que contiene el código de razón 2495 de IBM WebSphere MQ (MQRC_MODULE_NOT_FOUND) si:

   - Las clases IBM WebSphere MQ para la aplicación JMS se ejecutan en un entorno de ejecución Java de 32 bits, y se ha configurado un entorno de 64 bits para las clases IBM WebSphere MQ para JMS, ya que el entorno de ejecución Java de 32 bits no puede cargar la biblioteca nativa Java de 64 bits.
   - Las clases IBM WebSphere MQ para la aplicación JMS se ejecutan en un entorno de ejecución Java de 64 bits y se ha configurado un entorno de 32 bits para las clases IBM WebSphere MQ para JMS, ya que el entorno de ejecución Java de 64 bits no puede cargar la biblioteca nativa Java de 32 bits.

El archivo de configuración IBM WebSphere MQ classes for JMS

Un archivo de configuración de WebSphere MQ classes for JMS especifica las propiedades que se utilizan para configurar WebSphere MQ classes for JMS.

El formato de un archivo de configuración de WebSphere MQ para JMS 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 WebSphere MQ para 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 WebSphere MQ para JMS. 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 de las clases de WebSphere MQ para el archivo de configuración JMS. 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 las clases WebSphere MQ para el archivo de configuración JMS como el archivo D:\mydir\mjms.config en el sistema Windows local.

Cuando se inicia una aplicación, WebSphere MQ classes for JMS 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, WebSphere MQ classes for JMS 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 WebSphere MQ para JMS se puede utilizar con cualquiera de los transportes soportados entre una aplicación y un gestor de colas o intermediario.

Tenga en cuenta que no puede especificar el rastreo de inicio estableciendo una propiedad en el archivo de configuración de WebSphere MQ classes for JMS. Sólo puede especificar el rastreo de inicio estableciendo una propiedad del sistema en el mandato **java** , tal como se muestra en el ejemplo siguiente:

```
java -Dcom.ibm.msg.client.commonservices.trace.startup=true
     -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
      MyAppClass
```

Alteración temporal de las propiedades especificadas en un archivo de configuración de cliente MQI de WebSphere MQ

Un archivo de configuración de cliente MQI WebSphere MQ también puede especificar propiedades que se utilizan para configurar clases WebSphere MQ para JMS. Sin embargo, las propiedades especificadas en un archivo de configuración de cliente MQI de WebSphere MQ sólo se aplican cuando una aplicación se conecta a un gestor de colas en modalidad de cliente.

Si es necesario, puede alterar temporalmente cualquier atributo en un archivo de configuración de cliente MQI de WebSphere MQ especificándolo como una propiedad en un archivo de configuración de WebSphere MQ para JMS. Para alterar temporalmente un atributo en un archivo de configuración de cliente MQI WebSphere MQ , utilice una entrada con el formato siguiente en el archivo de configuración WebSphere MQ classes for JMS:

```
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 cliente MQI de WebSphere MQ que contiene el atributo

propName
El nombre del atributo tal como se especifica en el archivo de configuración del cliente MQI de WebSphere MQ

propValue
El valor de la propiedad que altera temporalmente el valor del atributo especificado en el archivo de configuración de cliente MQI WebSphere MQ

Como alternativa, puede alterar temporalmente un atributo en un archivo de configuración de cliente MQI de WebSphere MQ especificando la propiedad como propiedad del sistema en el mandato **java** . Utilice el formato anterior para especificar el atributo como propiedad del sistema.

Sólo los atributos siguientes en un archivo de configuración de cliente MQI de WebSphere MQ son relevantes para las clases de WebSphere MQ para JMS. 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	Atributo
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

IBM WebSphere MQ classes for JMS y herramientas de gestión de software

Herramientas de gestión de software tales como Apache Maven se pueden usar con las IBM WebSphere MQ classes for JMS.

Muchas organizaciones de desarrollo de gran tamaño utilizan estas herramientas para gestionar de forma centralizada los repositorios de bibliotecas de terceros.

Las IBM WebSphere MQ classes for JMS constan de una serie de archivos JAR. Al desarrollar aplicaciones de lenguaje Java utilizando esta API, es necesaria una instalación de un SupportPac de servidor, cliente o cliente de IBM WebSphere MQ 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 WebSphere 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 WebSphere MQ.
- El usuario es responsable de actualizar el repositorio con cualquier mantenimiento proporcionado por el soporte de IBM .

Para IBM WebSphere MQ Version 7.5, es necesario instalar los siguientes archivos JAR en el repositorio:

- com.ibm.mqjms.jar.
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- CL3Export.jar es necesario si está utilizando IBM WebSphere MQ classes for JMS.
- CL3Nonexport.jar es necesario si utiliza IBM WebSphere MQ classes for JMS.
- jndi.jar es necesario si utiliza IBM WebSphere MQ classes for JMS.
- ldap.jar es necesario si utiliza IBM WebSphere MQ classes for JMS.
- rmm.jar es necesario si está utilizando IBM WebSphere MQ classes for JMS.
- dhbcore.jar es necesario si está utilizando IBM WebSphere MQ classes for JMS.

- jms.jar es necesario si utiliza IBM WebSphere MQ classes for JMS.
- fscontext.jar es necesario si está utilizando IBM WebSphere MQ classes for JMS y accediendo a objetos administrados JMS almacenados en un contexto JNDI de sistema de archivos.
- providerutil.jar si está utilizando IBM WebSphere MQ classes for JMS y accediendo a objetos administrados JMS que están almacenados en un contexto JNDI de sistema de archivos.

Ejecución de clases de WebSphere MQ para aplicaciones JMS bajo el gestor de seguridad de Java

Las clases de WebSphere MQ para JMS se pueden ejecutar con el gestor de seguridad de Java habilitado. Para ejecutar aplicaciones correctamente con el gestor de seguridad habilitado, debe configurar la máquina virtual Java (JVM) con un archivo de configuración de políticas adecuado.

La forma más sencilla de hacerlo es cambiar el archivo de configuración de políticas proporcionado con el JRE. En la mayoría de los sistemas, este archivo se almacena en la vía de acceso `lib/security/java.policy`, relativa al directorio JRE. Puede editar el archivo de configuración de políticas utilizando el editor que prefiera o el programa `policytool` proporcionado con el JRE.

Importante: V 7.5.0.8 Siempre que ha sido posible, el término *lista de elementos permitidos* ha sustituido el término *lista blanca*. Una excepción son los siguientes nombres de propiedad del sistema Java .

Si utiliza el mecanismo del gestor de seguridad de Java 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.
- PropertyPermission (lectura) en las propiedades com.ibm.mq.jms.whitelist, com.ibm.mq.jms.whitelist.discover y com.ibm.mq.jms.whitelist.mode.

La lista de elementos permitidos ClassName está soportada con APAR IT14385 y IBM WebSphere MQ Version 7.5.0, Fixpack 8. Para obtener más información, consulte “Lista de elementos permitidos ClassName en JMS ObjectMessage” en la página 745.

A continuación se muestra un ejemplo de dos entradas en un archivo de configuración de políticas que permiten que WebSphere MQ classes for JMS se ejecute correctamente bajo el gestor de seguridad predeterminado:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "console.encoding","read";
  permission java.lang.RuntimePermission "setContextClassLoader";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.*","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
```

```
  permission java.util.logging.LoggingPermission "control";
  //Wherever trace output is expected
  permission java.io.FilePermission "/tmp/*","read,write";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
```

Notas:

- *MQ_INSTALLATION_PATH* representa el directorio de alto nivel en el que está instalado WebSphere MQ.
- La primera sentencia `grant` contiene los permisos necesarios para las clases WebSphere MQ para JMS, y la segunda sentencia `grant` contiene los permisos necesarios para una aplicación WebSphere MQ para JMS.
- Para permitir que WebSphere MQ classes for JMS acceda a los archivos JAR (Java Archive) de una aplicación, añada el permiso siguiente a la primera sentencia `grant` :

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- Para utilizar estas sentencias `grant` en el archivo de configuración de políticas, es posible que tenga que modificar los nombres de vía de acceso en función de dónde haya instalado WebSphere MQ classes for JMS y dónde almacene las aplicaciones.
- Las aplicaciones de ejemplo proporcionadas con WebSphere MQ classes for JMS, y los scripts para ejecutarlas, no habilitan el gestor de seguridad.

V 7.5.0.8 Lista de elementos permitidos ClassName en JMS ObjectMessage

V 7.5.0.8 En WebSphere MQ classes for JMS, el soporte para la lista de elementos permitidos de clases en la implementación de la interfaz JMS ObjectMessage 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 Java.

Nota: Siempre que ha sido posible, el término *lista de elementos permitidos* ha sustituido el término *lista blanca*. Una excepción son los nombres de propiedad del sistema Java mencionados en este tema.

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 Java Message Service (JMS) 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.

Lista de elementos permitidos en WebSphere MQ classes for JMS

Con APAR IT14385 y IBM WebSphere MQ Version 7.5.0, Fixpack 8, WebSphere MQ classes for JMS da soporte a la lista de permitidas de clases en la implementación de la interfaz ObjectMessage de JMS. La lista de elementos permitidos define qué clases Java se pueden serializar con ObjectMessage.setObject() y deserializar con ObjectMessage.getObject().

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.

Producción de la lista de elementos permitidos

Importante: Las clases WebSphere MQ para JMS 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 aplicación y IBM WebSphere MQ no puede adelantarse a ello.

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.whitelist
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 un gestor de seguridad de Java , debe asegurarse de que las clases WebSphere MQ para archivos JAR JMS tengan acceso de lectura y escritura a este archivo.

com.ibm.mq.jms.whitelist.discover

- 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 no se ha especificado la lista de elementos permitidos utilizando la propiedad com.ibm.mq.jms.whitelist, el mecanismo de lista de elementos permitidos 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.whitelist.mode
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: Es posible que el archivo necesite conversión si lo mueve entre sistemas heterogéneos.
- 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 UNIX o 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 graba una entrada en el archivo de registro JMS de IBM WebSphere MQ 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 IBM WebSphere MQ Advanced Message Security (AMS) puede añadir seguridad adicional al garantizar que los mensajes proceden de orígenes de confianza.

Si utiliza el mecanismo del gestor de seguridad de Java 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.
- PropertyPermission (lectura) en las propiedades com.ibm.mq.jms.whitelist, com.ibm.mq.jms.whitelist.discover y com.ibm.mq.jms.whitelist.mode.

Conceptos relacionados

"Ejecución de clases de WebSphere MQ para aplicaciones JMS bajo el gestor de seguridad de Java" en la página 744

Las clases de WebSphere MQ para JMS se pueden ejecutar con el gestor de seguridad de Java habilitado. Para ejecutar aplicaciones correctamente con el gestor de seguridad habilitado, debe configurar la máquina virtual Java (JVM) con un archivo de configuración de políticas adecuado.

El adaptador de recursos de IBM WebSphere MQ

El adaptador de recursos permite que las aplicaciones que se ejecutan en un servidor de aplicaciones accedan a los recursos de IBM WebSphere MQ . El adaptador de recursos da soporte a la comunicación de entrada y de salida.

La JCA (Java Platform, Enterprise Edition (Java EE) Connector Architecture) proporciona una forma estándar de conectar aplicaciones que se ejecutan en un entorno Java EE a un EIS (Enterprise Information System) como, por ejemplo, IBM WebSphere MQ o Db2. El adaptador de recursos de IBM WebSphere MQ implementa las interfaces JCA 1.5 y contiene IBM WebSphere MQ classes for JMS. Permite que las aplicaciones 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 WebSphere MQ . El adaptador de recursos soporta tanto el dominio punto a punto como el dominio de publicación/suscripción.

El adaptador de recursos de IBM WebSphere 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, a continuación, envía mensajes JMS a los destinos JMS y recibe mensajes JMS de los destinos JMS de forma síncrona.

Comunicación de entrada
: Un mensaje JMS que llega a un destino JMS se entrega a un MDB, que procesa el mensaje de forma asíncrona.

Para obtener más información acerca de la IBM WebSphere MQ classes for JMS, consulte "Utilización de clases de WebSphere MQ para JMS" en la página 729.

El adaptador de recursos también contiene el IBM WebSphere MQ classes for Java. Las clases están disponibles automáticamente para las 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 IBM WebSphere MQ classes for Java al acceder a los recursos de un gestor de colas IBM WebSphere MQ . Para obtener más información sobre la IBM WebSphere MQ classes for Java, consulte "Utilización de clases de WebSphere MQ para Java" en la página 666.

El uso de IBM WebSphere MQ classes for Java en un entorno Java EE está soportado con restricciones. Para obtener información sobre estas restricciones, consulte “Ejecución de clases IBM WebSphere MQ para aplicaciones Java en la plataforma Java Enterprise Edition” en la página 727.

Otra documentación necesaria para dar soporte a un adaptador de recursos JCA

Consulte la documentación del servidor de aplicaciones para obtener información sobre cómo configurar un adaptador de recursos JCA.

Cada servidor de aplicaciones proporciona su propio conjunto de interfaces de administración. Algunos servidores de aplicaciones proporcionan interfaces gráficas de usuario para definir recursos JCA, pero otros requieren que el administrador escriba planes de despliegue XML. Por lo tanto, está fuera del ámbito de esta documentación proporcionar información sobre cómo configurar el adaptador de recursos WebSphere MQ para cada servidor de aplicaciones. Esta documentación se centra únicamente en lo que necesita configurar. Consulte la documentación proporcionada con el servidor de aplicaciones para obtener información sobre cómo configurar un adaptador de recursos JCA.

Para comprender esta documentación, debe estar familiarizado con JMS y las clases de WebSphere MQ para JMS. Muchas de las propiedades utilizadas para configurar el adaptador de recursos de WebSphere MQ son equivalentes a las propiedades de las clases de WebSphere MQ para objetos JMS y tienen la misma función.

Instalación del adaptador de recursos WebSphere MQ

El adaptador de recursos WebSphere 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.

El adaptador de recursos WebSphere MQ se proporciona como un archivo de archivado de recursos (RAR) denominado wmq.jmsra.rar. Este archivo se instala con WebSphere MQ classes for JMS en el directorio que se muestra en la Tabla 94 en la página 749.

Tabla 94. Directorio donde reside wmq.jmsra.rar para cada plataforma

Plataforma	**Directorio**
AIX, HP-UX, Linux y Solaris	*MQ_INSTALLATION_PATH* /java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH* \java\lib\jca
MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.	

El archivo RAR contiene clases WebSphere MQ para JMS y la implementación WebSphere MQ de las interfaces JCA.

Debe instalar el archivo RAR del adaptador de recursos de WebSphere MQ en el servidor de aplicaciones, pero la forma en que lo haga depende del servidor de aplicaciones. Consulte la documentación del servidor de aplicaciones para obtener información sobre cómo instalar un archivo RAR de adaptador de recursos.

Para conexiones de enlaces en sistemas UNIX and Linux , debe asegurarse de que el directorio que contiene las bibliotecas JNI (Java Native Interface) esté en la vía de acceso del sistema. Para ver la ubicación de este directorio, que también contiene las clases WebSphere MQ para bibliotecas JMS, consulte Tabla 93 en la página 739. En Windows, este directorio se añade automáticamente a la vía de acceso del sistema durante la instalación de WebSphere MQ classes for JMS.

Las transacciones están soportadas tanto en la modalidad de cliente como en la modalidad de enlaces.

El adaptador de recursos de WebSphere MQ y la versión de las clases de WebSphere MQ para JMS utilizadas por el adaptador de recursos deben estar en el mismo nivel de release.

WebSphere Application Server y el adaptador de recursos WebSphere MQ
No utilice el adaptador de recursos WebSphere MQ con WebSphere Application Server Versión 6. WebSphere Application Server, V7, incluye una versión del adaptador de recursos WebSphere MQ V7 .

No utilice el adaptador de recursos WebSphere MQ dentro de WebSphere Application Server, V6. Para acceder a los recursos de un gestor de colas WebSphere MQ desde dentro de WebSphere Application Server desde una aplicación JMS, utilice el proveedor de mensajería WebSphere MQ . El proveedor de mensajería de WebSphere MQ contiene una versión de las clases de WebSphere MQ para JMS.

WebSphere Application Server, V7, incluye una versión del adaptador de recursos WebSphere MQ V7 .

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?).

WebSphere Application Server Liberty y el adaptador de recursos de IBM WebSphere MQ
El adaptador de recursos de IBM WebSphere MQ Version 7.5 se puede instalar en WebSphere Application Server Liberty Versión 8.5.5, Fixpack 2 o posterior, utilizando la característica wmqJmsClient-1.1 . De forma alternativa, sujeto a algunas 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 Version 7.5 cuando se utiliza la característica wmqJmsClient-1.1 y también cuando se utiliza el soporte JCA genérico:

- Los IBM WebSphere MQ classes for Java no están soportados en Liberty. No se deben utilizar con la característica de mensajería de IBM WebSphere MQ Liberty ni con el soporte de JCA genérico. Para obtener más información, consulte Utilización de interfaces Java de WebSphere MQ en entornos J2EE/JEE.
- El adaptador de recursos de IBM WebSphere MQ tiene un tipo de transporte de BINDINGS_THEN_CLIENT. Este tipo de transporte no está soportado en la característica de mensajería de Liberty de IBM WebSphere MQ .
- La característica IBM WebSphere MQ Advanced Message Security (IBM WebSphere MQ AMS) no se incluye en la característica de mensajería de Liberty de IBM WebSphere MQ .

El adaptador de recursos de IBM WebSphere MQ Version 7.5 no se puede utilizar con la característica wmqJmsClient-2.0 .

Configuración del adaptador de recursos WebSphere MQ

Para configurar el adaptador de recursos WebSphere MQ , debe definir varios recursos JCA y propiedades del sistema.

Defina recursos JCA en las categorías siguientes:

- Las propiedades del objeto ResourceAdapter , que representan las propiedades globales del adaptador de recursos, como el nivel de rastreo de diagnóstico. Estas propiedades se describen en “Configuración del objeto ResourceAdapter” en la página 751.
- Las propiedades de un objeto ActivationSpec , que determinan cómo se activa un MDB para la comunicación de entrada. Estas propiedades se describen en “Configuración del adaptador de recursos para la comunicación de entrada” en la página 753.
- Las propiedades de un objeto ConnectionFactory , que el servidor de aplicaciones utiliza para crear un objeto ConnectionFactory JMS para la comunicación de salida. Estas propiedades se describen en “Configuración del adaptador de recursos para la comunicación de salida” en la página 767.
- Las propiedades de un objeto de destino administrado, que el servidor de aplicaciones utiliza para crear un objeto de cola JMS u objeto de tema JMS para la comunicación de salida. Estas propiedades también se describen en “Configuración del adaptador de recursos para la comunicación de salida” en la página 767.

El archivo RAR del adaptador de recursos de WebSphere MQ contiene un archivo denominado 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://java.sun.com/xml/ns/j2ee/connector_1_5.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. Por ejemplo, WebSphere Application Server Community Edition requiere un plan de despliegue denominado geronimo-ra.xml.

Si utiliza SSL (Secure Sockets Layer), especifique las ubicaciones del archivo de almacén de claves y del archivo de almacén de confianza como propiedades del 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 toda la JVM y, por lo tanto, pueden afectar al servidor de aplicaciones si otras aplicaciones, que se ejecutan en el servidor de aplicaciones, utilizan conexiones SSL. También es posible que el servidor de aplicaciones restablezca estas propiedades en valores distintos. Para obtener más información sobre cómo utilizar SSL con las clases WebSphere MQ para JMS, consulte “Utilización de SSL (Secure Sockets Layer) con clases WebSphere MQ para JMS” en la página 925.

Se proporciona un programa de prueba de verificación de instalación (IVT) con el adaptador de recursos WebSphere MQ , pero debe configurar el adaptador de recursos para poder ejecutar el programa. Para obtener más información sobre lo que debe configurar para poder ejecutar el programa IVT, consulte “El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ” en la página 799.

Los registros, avisos y mensajes de error del adaptador de recursos utilizan el mismo mecanismo que las clases IBM WebSphere MQ para JMS, para obtener detalles, consulte “Registro yIBM WebSphere MQ classes for JMS” en la página 813. Para WebSphere Application Server, estos mensajes se redirigen automáticamente al registro de salida del servidor de aplicaciones. Para otros servidores de aplicaciones, como WAS CE y JBoss, estos irán, de forma predeterminada, a un archivo denominado mqjms.log. Para configurar el adaptador de recursos para registrar adicionalmente los mensajes de aviso en el registro de salida estándar de los servidores de aplicaciones, establezca la siguiente propiedad del sistema JVM para el servidor de aplicaciones:

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

Para obtener detalles sobre cómo establecer una propiedad del sistema JVM, consulte la documentación del servidor de aplicaciones.

Configuración del objeto ResourceAdapter
El objeto ResourceAdapter encapsula las propiedades globales del adaptador de recursos de WebSphere MQ . Defina estas propiedades utilizando los recursos del adaptador de recursos.

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 manera en que define estas propiedades depende de las interfaces de administración proporcionadas por el servidor de aplicaciones.

Para obtener más información sobre la definición de propiedades asociadas con el rastreo de diagnóstico, consulte Rastreo del adaptador de recursos de IBM WebSphere MQ.

El adaptador de recursos gestiona una agrupación de conexiones internas de conexiones JMS que se utilizan para entregar mensajes a los MDB. Tabla 95 en la página 752 lista las propiedades del objeto ResourceAdapter que están asociadas con la agrupación de conexiones.

Tabla 95. Propiedades del objeto ResourceAdapter que están asociadas con la agrupación de conexiones			
Nombre de la propiedad	**Tipo**	**Valor predeterminado**	**Descripción**
maxConnections	Cadena	50	El número máximo de conexiones a un gestor de colas de WebSphere MQ y el número máximo de MDB desplegados.
connectionConcurrency	Cadena	1	El número máximo de MDBs para compartir una conexión JMS. El uso compartido de conexiones no es posible y el valor de esta propiedad es siempre 1.
reconnectionRetryCount	Cadena	5	El número máximo de intentos realizados por el adaptador de recursos para volver a conectarse a un gestor de colas de WebSphere MQ si falla una conexión.
reconnectionRetryInterval	Cadena	300000	El tiempo, en milisegundos, que el adaptador de recursos espera antes de intentar volver a conectarse a un gestor de colas de WebSphere MQ .
startupRetryCount	Cadena	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	Cadena	30000	Tiempo de inactividad predeterminado entre los intentos de conexión durante el arranque (en milisegundos).

Cuando se despliega un MDB en el servidor de aplicaciones, se crea una nueva conexión JMS y se inicia una conversación con el gestor de colas, siempre que no se supere 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 de intervalo reconnectionRetryy reconnectionRetrycontrolan el comportamiento del adaptador de recursos cuando fallan las conexiones con un gestor de colas WebSphere 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 UNIX 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 WebSphere MQ sólo se ejecutan en el

contenedor de cliente. En esta situación, puede establecer las propiedades para el rastreo de diagnóstico como propiedades del sistema JVM (Java Virtual Machine). 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 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 de unidad de mensajes (MDB) recibe mensajes JMS de una cola WebSphere 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 WebSphere MQ
- Propiedades que se utilizan para crear un consumidor de conexión JMS que entrega mensajes de forma asíncrona a medida que llegan a 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.

La Tabla 96 en la página 753 lista las propiedades de un objeto ActivationSpec que se utilizan para crear una conexión JMS con un gestor de colas WebSphere MQ .

Tabla 96. 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 **Conexiones de aplicación** de IBM WebSphere MQ Explorer (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

Tabla 96. 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
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
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 WebSphere MQ . O para utilizar un intermediario de WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker o WebSphere Business Integration Message Broker en modalidad de compatibilidad. Este valor es el valor predeterminado si TRANSPORT se establece en BIND o CLIENT. • **V2** -Para utilizar un intermediario de WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker o WebSphere Business Integration Message Broker 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** • Un identificador de juego de caracteres codificado 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

Tabla 96. 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
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
cloneSupport	Serie	• **DISABLED** - Sólo se puede ejecutar una instancia de un suscriptor de temas duradero simultáneamente. • ENABLED-Dos o más instancias del mismo suscriptor de tema duradero pueden ejecutarse simultáneamente, pero cada instancia debe ejecutarse en una máquina virtual Java (JVM) independiente.	Determina si dos o más instancias del mismo suscriptor de temas duradero pueden ejecutarse simultáneamente
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 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.

Tabla 96. 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
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
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. Las propiedades **hostname** y **port** se reemplazan por la propiedad **connectionNameList** cuando se especifica.
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

Tabla 96. 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
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.
messageSelection [1]	Serie	• **CLIENT** • BROKER	Determina si la selección de mensajes la realizan las clases WebSphere MQ para JMS o 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. Las propiedades **hostname** y **port** se reemplazan por la propiedad **connectionNameList** cuando se especifica.

Tabla 96. 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** • Una serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa las clases WebSphere MQ para la interfaz Java, MQReceiveExit	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 96. 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 del dominio punto a punto utiliza un selector de mensajes para seleccionar qué mensajes desea recibir, WebSphere MQ classes for JMS busca en la cola WebSphere MQ los mensajes adecuados en la secuencia determinada por el atributo *MsgDeliverySequence* de la cola. Cuando WebSphere MQ classes for JMS encuentra un mensaje adecuado y lo entrega al consumidor, WebSphere MQ classes for JMS reanuda la búsqueda del siguiente mensaje adecuado desde su posición actual en la cola. WebSphere MQ classes for JMS continúa buscando en la cola de esta forma hasta que alcanza el final de la cola, o hasta que el intervalo de tiempo en milisegundos, determinado por el valor de esta propiedad, ha caducado. En cada caso, WebSphere 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** • El nombre completo de una clase que implementa las clases de WebSphere MQ para la interfaz Java, MQSecurityExit	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

Tabla 96. 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
sendExit [3]	Serie	• **null** • Una serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa las clases WebSphere MQ para la interfaz Java, MQSendExit	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 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.	Si una conexión de cliente puede compartir su socket con otras conexiones JMS de nivel superior del mismo proceso 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.	Los servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados (CRL) para su uso en una conexión SSL
sslCipherSuite	Serie	• **null** • El nombre de un CipherSuite	La CipherSuite que se debe utilizar para una conexión SSL
sslFipsRequired[2]	Boolean	• **false** • true	Si una conexión SSL debe utilizar una CipherSuite soportada por el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS)

Tabla 96. 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 SSL, 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	El número total de bytes enviados y recibidos por una conexión SSL antes de que se renegocien las claves secretas utilizadas por SSL
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 relevante si subscriptionStore tiene el valor QUEUE.
subscriptionStore [1]	Serie	• **Intermediario** • MIGRATE • COLA	Determina dónde WebSphere MQ classes for JMS almacena datos persistentes sobre suscripciones activas
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 realizar una conexión en modalidad de enlaces. Si este intento de conexión falla, el adaptador de recursos intenta realizar una conexión en modalidad de cliente.

Tabla 96. 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**
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
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 7.0 de las clases WebSphere MQ para JMS. No afecta a una aplicación conectada a un gestor de colas de la versión 7.0 a menos que la propiedad providerVersion esté establecida en un número de versión inferior a 7.
2. Para conocer información importante sobre la utilización de la propiedad sslFipsRequired, consulte "Limitaciones del adaptador de recursos de IBM WebSphere MQ" en la página 787.
3. Para obtener información sobre cómo configurar el adaptador de recursos para que pueda localizar una salida, consulte "Configuración de IBM WebSphere MQ classes for JMS para utilizar salidas de canal" en la página 931.

Tabla 97 en la página 762 lista las propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexiones JMS.

Tabla 97. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión 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.
destinationType	Serie	• javax.jms.Queue • javax.jms.Topic	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

Tabla 97. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
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 sólo es válido cuando la aplicación se ejecuta en WebSphere Application Server versión 7 o posterior.
nonASFRollbackEnabled	Boolean	• **false** - El mensaje se consume incluso si el MDB falla • true-La anomalía en el MDB hace que el mensaje se retrotraiga a la cola.	Indica si la entrega de mensajes está dentro de un punto de sincronización de WebSphere MQ si el MDB no es transaccional. Se ignora si el MDB es transaccional o si nonASFTimeout se establece 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.	Determina si el MDB puede utilizar la lectura anticipada para obtener mensajes no persistentes del destino y colocarlos en un almacenamiento intermedio interno antes de recibir los mensajes.

Tabla 97. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión JMS (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
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.
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 ignora 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.
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 97. Propiedades de un objeto ActivationSpec que se utilizan para crear un consumidor de conexión 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 un tema de WebSphere MQ . • true - La propiedad denominada destination se interpreta como el nombre de un objeto javax.jms.Queue o javax.jms.Topic en el espacio de nombres JNDI del servidor de aplicaciones.	Determina cómo se interpreta el valor de la propiedad denominada destination

Las propiedades ActivationSpec denominadas destination y destinationType deben definirse explícitamente. Las demás propiedades son opcionales.

Un objeto ActivationSpec puede tener propiedades en conflicto. Por ejemplo, puede especificar propiedades SSL para una conexión en 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 tal como lo determina 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 las propiedades denominadas ccdtURL y el canal están ambos definidos, se genera una excepción de excepción InvalidProperty. Sin embargo, si define sólo la propiedad ccdtURL , dejando la propiedad llamada 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.

La mayoría de las propiedades de un objeto ActivationSpec son equivalentes a las propiedades de WebSphere MQ clases para objetos JMS o parámetros de WebSphere MQ clases para métodos JMS. Sin embargo, tres propiedades de ajuste y una propiedad de usabilidad no tienen equivalentes en las clases WebSphere MQ para JMS:

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 97 en la página 762.

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:

```
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 WebSphere MQ” en la página 749.

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.

Manejo de mensajes con formato incorrecto de entrada en el adaptador de recursos

En algunas circunstancias, un mensaje entregado a un MDB puede retrotraerse a una cola de WebSphere 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 WebSphere MQ para que las clases WebSphere

MQ para JMS transfieran automáticamente un mensaje con formato incorrecto a otra cola para una investigación adicional o descarte el mensaje.

Para obtener detalles sobre cómo manejar mensajes no entregables, consulte “Manejo de mensajes no entregables en IBM WebSphere MQ classes for JMS” en la página 908.

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 UNIX, Linux y Windows

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.

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:

```
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 HTTP GET, recupera un objeto ConnectionFactory y un objeto Queue del espacio de nombres JNDI y utiliza los objetos para enviar un mensaje a una cola WebSphere MQ . A continuación, el servlet recibe el mensaje que ha enviado.

Para configurar la comunicación de salida, defina los recursos 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 JMS o un objeto de tema 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.

La Tabla 98 en la página 768 lista las propiedades de un objeto ConnectionFactory.

Tabla 98. 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 WebSphere MQ Explorer (donde el campo se denomina **App name**).
brokerCCSubQueue [1]	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.
brokerControlQueue [1]	Serie	• **SYSTEM.BROKER.CONTROL.QUEUE** • El nombre de una cola	El nombre de la cola de control de intermediario.
brokerPubQueue[1]	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).
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 a partir de la cual un consumidor de mensajes no duradero recibe mensajes. Consulte la propiedad BROKERSUBQ para obtener más información.

Tabla 98. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
brokerVersion[1]	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 WebSphere MQ . O para utilizar un intermediario de IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker o WebSphere Business Integration Message Broker en modalidad de compatibilidad. Este valor es el valor predeterminado si TRANSPORT se establece en BIND o CLIENT. • **V2** -Para utilizar un intermediario de IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker o WebSphere Business Integration Message Broker 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 [1]	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 98. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
cleanupLevel[1]	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 98. 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. Las propiedades **hostname** y **port** se reemplazan por la propiedad **connectionNameList** cuando se especifica.

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 [1]	Serie	• **CLIENT** • BROKER	Determina si la selección de mensajes la realizan las clases IBM WebSphere MQ para JMS o 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 98. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
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. Las propiedades **hostname** y **port** se reemplazan por la propiedad **connectionNameList** cuando se especifica.
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.
pubAckInterval[1]	int	• **25** • Un entero positivo	Número de mensajes publicados por un publicador antes de que IBM WebSphere 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 98. 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 WebSphere 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[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, WebSphere MQ classes for JMS busca en la cola IBM WebSphere MQ los mensajes adecuados en la secuencia determinada por el atributo *MsgDeliverySequence* de la cola. Cuando WebSphere MQ classes for JMS encuentra un mensaje adecuado y lo entrega al consumidor, WebSphere MQ classes for JMS reanuda la búsqueda del siguiente mensaje adecuado desde su posición actual en la cola. WebSphere MQ classes for JMS continúa buscando en la cola de esta forma hasta que alcanza el final de la cola, o hasta que el intervalo de tiempo en milisegundos, determinado por el valor de esta propiedad, ha caducado. En cada caso, WebSphere 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** • El nombre completo de una clase que implementa las clases de WebSphere MQ para la interfaz Java, MQSecurityExit	Identifica un programa de salida de seguridad de canal.

Tabla 98. 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	El número de llamadas de envío que se permiten entre la comprobación de errores de colocación asíncrona, dentro de una única sesión JMS sin transacción.
sendExit [3]	Serie	• **null** • Una serie que consta de uno o más elementos separados por comas, donde cada elemento es el nombre completo de una clase que implementa las clases WebSphere MQ para la interfaz Java, MQSendExit	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.	Si una conexión de cliente puede compartir su socket con otras conexiones JMS de nivel superior del mismo proceso 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.

Tabla 98. 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.	Los servidores LDAP (Lightweight Directory Access Protocol) que contienen listas de revocación de certificados (CRL) para su uso en una conexión SSL.
sslCipherSuite	Serie	• **null** • El nombre de un CipherSuite	La CipherSuite que se debe utilizar para una conexión SSL.
sslFipsRequired[2]	Boolean	• **false** • true	Si una conexión SSL debe utilizar una CipherSuite soportada por el proveedor JSSE FIPS (IBMJSSEFIPS) de IBM Java .
sslPeerName	Serie	• **null** • Una plantilla para nombres distinguidos	Para una conexión SSL, 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 SSL antes de que se renegocien las claves secretas utilizadas por SSL.
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.
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 relevante si SUBSTORE tiene el valor QUEUE.
subscriptionStore[1]	Serie	• **Intermediario** • MIGRATE • COLA	Determina dónde WebSphere MQ classes for JMS almacena datos persistentes sobre suscripciones activas.

Tabla 98. Propiedades de un objeto ConnectionFactory (continuación)

Nombre de la propiedad	Tipo	Valores válidos (los valores predeterminados se muestran en negrita)	Descripción
targetClientMatching	Boolean	• **true** • falso	Si un mensaje de respuesta, enviado a la cola identificada por el campo de cabecera JMSReplyTo de un mensaje de entrada, tiene una cabecera MQRFH2 sólo si el mensaje de entrada tiene una cabecera MQRFH2 .

Tabla 98. 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	El nombre de la cola modelo a partir de la cual se crean las colas temporales JMS. Utilice SYSTEM.DEFAULT.MODEL.QUEUE si se cumplen las dos condiciones 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. Tenga en cuenta que SYSTEM.DEFAULT.MODEL.QUEUE sólo puede ser abierto por una aplicación a la 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 cola modelo nueva 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 de modelo, establezca la propiedad temporaryModel en el nombre de la nueva cola de modelo.

Tabla 98. 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 WebSphere 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 WebSphere 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, WebSphere 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 WebSphere MQ.
tempTopicPrefix	Serie	Cualquier serie no nula que conste solamente de caracteres válidos para una serie de tema de IBM WebSphere 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*". Especificar un TEMPTOPICPREFIX no vacío permite definir colas de modelo específicas para crear las colas gestionadas para suscriptores a temas temporales creados bajo esta conexión.
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 realizar una conexión en modalidad de enlaces. Si este intento de conexión falla, el adaptador de recursos intenta entonces establecer una conexión en la modalidad de cliente.
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.

Tabla 98. Propiedades de un objeto ConnectionFactory (continuación)			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
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. Esta propiedad se puede utilizar con la versión 7.0 de IBM WebSphere MQ classes for JMS pero no afecta a una aplicación conectada a un gestor de colas de la versión 7.0 a menos que la propiedad providerVersion se establezca en un número de versión menor que 7.
2. Para conocer información importante sobre la utilización de la propiedad sslFipsRequired, consulte "Limitaciones del adaptador de recursos de IBM WebSphere MQ" en la página 787.
3. Para obtener información sobre cómo configurar el adaptador de recursos para que pueda localizar una salida, consulte "Configuración de IBM WebSphere MQ classes for JMS para utilizar salidas de canal" en la página 931.

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
```

La Tabla 99 en la página 779 lista las propiedades que son comunes a un objeto de cola y un objeto de tema.

Tabla 99. 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.

Tabla 99. 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
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 zSeries . NATIVE es equivalente a la serie NNN.	Representación de enteros binarios, enteros decimales empaquetados y números de coma flotante para el destino.
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 99. 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 viene determinada por el atributo *DefPersistence* de la cola WebSphere MQ . • PERS - El mensaje es persistente. • NON - El mensaje es no persistente • HIGH-La persistencia de un mensaje viene determinada por el atributo *NonPersistentMessageClass* de la cola de WebSphere MQ de acuerdo con la explicación de "Mensajes persistentes JMS" en la página 923.	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 viene determinada por el atributo *DefPriority* de la cola IBM WebSphere 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 99. 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 ignora 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 JMS. • MQ -El destino de un mensaje es una aplicación IBM WebSphere MQ no JMS.	Si el destino de un mensaje enviado al destino es una aplicación JMS. Un mensaje con un destino que es una aplicación JMS contiene una cabecera MQRFH2 .

La Tabla 100 en la página 782 lista las propiedades que son específicas de un objeto Queue.

Tabla 100. 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 WebSphere MQ subyacente.
baseQueueName	Serie	• **"" (serie vacía)** • El nombre de una cola	El nombre de la cola de IBM WebSphere MQ subyacente.

La Tabla 101 en la página 783 lista las propiedades que son específicas de un objeto Topic.

Tabla 101. 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[1]	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 [1]	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 WebSphere MQ Explorer para obtener más información.
brokerPubQueue[1]	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 de cola brokerPubdel objeto ConnectionFactory . Sin embargo, si no establece el valor de esta propiedad, se utiliza en su lugar el valor de la propiedad de cola brokerPubdel objeto ConnectionFactory .
brokerPubQueueManager [1]	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[1]	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 . Sin embargo, si no establece el valor de esta propiedad, en su lugar se utiliza el valor de la propiedad brokerVersion del objeto ConnectionFactory .

Tabla 101. Propiedades que son específicas de un objeto Topic (continuación)			
Nombre de la propiedad	**Tipo**	**Valores válidos (los valores predeterminados se muestran en negrita)**	**Descripción**
Nota: 1. Esta propiedad se puede utilizar con la versión 7.0 de IBM WebSphere MQ classes for JMS pero no afecta a una aplicación conectada a un gestor de colas de la versión 7.0 a menos que la propiedad providerVersion del objeto ConnectionFactory esté establecida en un número de versión menor que 7.			

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

Referencia relacionada

Federal Information Processing Standards (FIPS) para UNIX, Linux y Windows

V 7.5.0.9 *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 WebSphere 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 tradicional como en WebSphere Application Server 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 WebSphere 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 tradicional, utilice la consola de administración para definir la propiedad **targetClientMatching** como una propiedad personalizada en la especificación de activación IBM WebSphere 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 Application Server 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 JMS" en la página 896
En lugar de recuperar destinos como objetos administrados desde un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación JMS puede utilizar una sesión para crear destinos dinámicamente en tiempo de ejecución. Una aplicación puede utilizar un identificador uniforme de recursos (URI) para identificar una cola o un tema de WebSphere MQ y, opcionalmente, para especificar una o más propiedades de un objeto Queue o Topic.

"Configuración del adaptador de recursos para la comunicación de salida" en la página 767
Para configurar la comunicación de salida, defina las propiedades de un objeto ConnectionFactory y un objeto de destino administrado.

ASF y modalidad no ASF
La modalidad ASF (Application Server Facilities) es el método predeterminado mediante el cual el servicio de escucha de mensajes de WebSphere Application Server procesa los mensajes.

El servicio de escucha de mensajes tiene dos modalidades de operación, ASF (Application Server Facilities) y no ASF (Application Server Facilities):

- La modalidad ASF proporciona soporte transaccional y de simultaneidad para las aplicaciones. Para los beans de unidad de mensajes de publicación/suscripción, la modalidad ASF proporciona un mejor rendimiento y simultaneidad, porque en la modalidad no ASF el escucha es de una sola hebra.
- La modalidad no ASF se utiliza principalmente con proveedores de mensajería de terceros que no dan soporte a JMS ASF, que es una extensión opcional de la especificación JMS. La modalidad no ASF también es transaccional pero, debido a que la longitud de la vía de acceso es menor que para la modalidad ASF, normalmente proporciona un rendimiento mejorado.

Para habilitar la modalidad de operación no ASF para todos los escuchas de beans controlados por mensajes en el servidor de aplicaciones, establezca esta propiedad en un valor distinto de cero.

Nota:

La modalidad no ASF no se puede seleccionar en sistemas z/OS , por lo que no debe establecer un valor distinto de cero para esta propiedad en este caso.

Proceso de mensajes en modalidad ASF

En la modalidad ASF, las sesiones de servidor y las hebras sólo se asignan para el trabajo cuando se detecta un mensaje que es adecuado para el bean controlado por mensaje (MDB). El número de hebras

que un MDB puede procesar simultáneamente viene determinado por el valor de la propiedad **Maximum Sessions** para el puerto de escucha o la especificación de activación.

Proceso de mensajes en modalidad no ASF

En modalidad no ASF, las hebras están activas desde el momento en que se inicia el puerto de escucha o la especificación de activación. El número de hebras activas viene determinado por el valor especificado para la propiedad **Maximum Sessions** . El número de hebras especificadas en la propiedad **Maximum Sessions** están activas, independientemente del número de mensajes que estén disponibles para procesarse. Cada hebra activa es una conexión de red física individual.

IBM WebSphere MQ Versión 7.0 o posterior le permite tener hasta diez hebras que comparten una única conexión de red física.

Conceptos relacionados

IBM WebSphere MQ classes for JMS Application Server Facilities
En este tema se describe cómo WebSphere MQ classes for JMS implementa la clase ConnectionConsumer y la funcionalidad avanzada en la clase Session. También describe la función de una agrupación de sesiones de servidor.

Tareas relacionadas

Configuración de especificaciones de activación para modalidad no ASF
Las especificaciones de activación son la forma estandarizada de gestionar y configurar la relación entre un bean controlado por mensajes (MDB) que se ejecuta en WebSphere Application Server y un destino en IBM WebSphere MQ. Esta tarea explica cómo configurar WebSphere Application Server para utilizar la modalidad no ASF para procesar mensajes.

Información relacionada

Proceso de mensajes en modalidad ASF y en modalidad no ASF

Configuración de especificaciones de activación para la modalidad no ASF
Las especificaciones de activación son la forma estandarizada de gestionar y configurar la relación entre un bean controlado por mensajes (MDB) que se ejecuta en WebSphere Application Server y un destino en IBM WebSphere MQ. Esta tarea explica cómo configurar WebSphere Application Server para utilizar la modalidad no ASF para procesar mensajes.

Antes de empezar

La forma en que se definen las propiedades de una especificación de activación depende de las interfaces de administración proporcionadas por el servidor de aplicaciones. Esta tarea presupone que está utilizando WebSphere Application Server versión 7 o posterior como servidor de aplicaciones y IBM WebSphere MQ como proveedor de mensajería.

Nota:

La modalidad no ASF no se puede seleccionar en sistemas z/OS .

Acerca de esta tarea

Las propiedades de una especificación de activación determinan cómo un bean de unidad de mensajes (MDB) recibe mensajes JMS de una cola IBM WebSphere MQ . Para configurar la modalidad no ASF, defina las propiedades de una o más especificaciones de activación.

Hay varias configuraciones de IBM WebSphere MQ que puede utilizar en modalidad no ASF. Con las configuraciones siguientes, cada hebra utiliza una conexión de red física independiente:

- Un gestor de colas IBM WebSphere MQ Versión 7.x , utilizando una fábrica de conexiones que tiene la propiedad Versión de proveedor establecida en 6.
- Un gestor de colas IBM WebSphere MQ Versión 7.x , utilizando una fábrica de conexiones que tiene la propiedad Versión de proveedor establecida en 7 o no especificada, que se conecta a través de un canal IBM WebSphere MQ que tiene el parámetro **SHARECNV** (compartir conversaciones) establecido en 0.

Para configurar no ASF, establezca la propiedad ActivationSpec **NON.ASF.RECEIVE.TIMEOUT** en un entero positivo, que 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. Para obtener más detalles, consulte **Propiedades personalizadas del servicio de escucha de mensajes**.

Este parámetro sólo es válido cuando la aplicación se ejecuta en WebSphere Application Server versión 7 o posterior.

Procedimiento

1. Inicie la consola administrativa de WebSphere Application Server.
2. Visualice la página de valores del servicio de escucha:
   a) En el panel de navegación, seleccione **Servidores > Tipos de servidor > WebSphere Application Servers**.
   b) En el panel de contenido, pulse el nombre del servidor de aplicaciones.
   c) En **Comunicaciones**, pulse **Mensajería > Servicio de escucha de mensajes**.
3. Establezca la propiedad personalizada **NON.ASF.RECEIVE.TIMEOUT** como propiedades personalizadas del servicio de escucha de mensajes.
   a) Pulse **Propiedades personalizadas**.
   b) Pulse **Nuevo**.
   c) Especifique el nombre de la propiedad, **NON.ASF.RECEIVE.TIMEOUT**, en el campo **Nombre** .
   d) Especifique el valor que necesita en el campo **Valor** .
   e) Pulse **Aceptar**.
4. Guarde los cambios realizados en la configuración maestra.
5. Para activar la configuración cambiada, detenga y, a continuación, reinicie el servidor de aplicaciones.

Resultados

Ha configurado las propiedades del servicio de escucha de mensajes para que WebSphere Application Server utilice la modalidad no ASF.

Nota: Cuando se utiliza la modalidad no ASF, debe asegurarse de que deja una cantidad de tiempo suficiente para que se complete el proceso antes de que se alcance el tiempo de espera de vida total de la transacción, para evitar tiempos de espera de transacción no deseados. Para obtener más detalles, consulte **NON.ASF.RECEIVE.TIMEOUT** en la documentación del producto WebSphere Application Server .

Conceptos relacionados

“ASF y modalidad no ASF” en la página 785
La modalidad ASF (Application Server Facilities) es el método predeterminado mediante el cual el servicio de escucha de mensajes de WebSphere Application Server procesa los mensajes.

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.

Información relacionada

Beans controlados por mensajes

Servicio de escucha de mensajes

Proceso de mensajes en modalidad ASF y en modalidad no ASF

Cómo se procesan los mensajes en modalidad no ASF

Limitaciones del adaptador de recursos de IBM WebSphere MQ

Cuando se utiliza el adaptador de recursos de IBM WebSphere MQ , algunas características de IBM WebSphere MQ no están disponibles o están limitadas.

El adaptador de recursos de IBM WebSphere MQ tiene las limitaciones siguientes:

- El adaptador de recursos IBM WebSphere MQ está soportado en todas las plataformas IBM WebSphere MQ , excepto z/OS.
- El adaptador de recursos de IBM WebSphere MQ no da soporte a conexiones en tiempo real con un intermediario. Solo da soporte a conexiones con un gestor de colas IBM WebSphere MQ en modalidad de cliente o enlaces.
- El adaptador de recursos de IBM WebSphere MQ no da soporte a programas de salida de canal escritos en lenguajes que no sean Java.
- Mientras se ejecuta un servidor de aplicaciones, el valor de la propiedad necesaria sslFipsdebe ser true para todos los recursos JCA o false para todos los recursos JCA. Este es un requisito incluso si los recursos JCA no se utilizan simultáneamente. Si la propiedad sslFipsRequired tiene distintos valores para distintos recursos JCA, IBM WebSphere MQ emite el código de razón MQRC_UNSUPPORTED_CIPHER_SUITE, aunque no se esté utilizando una conexión SSL.
- 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 introducida en IBM WebSphere MQ Version 7.0.1 no está soportada para conexiones de salida cuando se ejecuta en un contenedor JEE (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 JEE, independientemente de la configuración de transacción o para uso no transaccional.

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 UNIX, Linux y Windows

Configuración posterior a la instalación para las clases de WebSphere MQ para aplicaciones JMS

Este tema le indica qué autorizaciones necesitan las clases WebSphere MQ para las aplicaciones 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 comprobar el archivo léame de WebSphere MQ . Podría contener información que reemplace la información de este tema.

Objetos utilizados por JMS que requieren autorización para usuarios no privilegiados

Los usuarios sin privilegios necesitan autorización otorgada para acceder a las colas utilizadas por JMS. Cada aplicación JMS necesita autorización para el gestor de colas con el que funciona.

Para obtener detalles sobre el control de acceso en IBM WebSphere MQ, consulte Configuración de la seguridad en sistemas Windows, UNIX and Linux .

Las clases WebSphere MQ para aplicaciones JMS necesitan autorización `connect` y `inq` para el 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 WebSphere MQ para JMS se conectan a un gestor de colas IBM WebSphere MQ en modalidad de migración de proveedor de mensajería de IBM WebSphere 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 de proveedor de mensajería de IBM WebSphere MQ , consulte Cuándo utilizar **`PROVIDERVERSION`**

Además, si las clases de WebSphere MQ para 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 de tema o fábrica TopicConnection. De forma predeterminada, esta cola es SYSTEM.BROKER.DEFAULT.STREAM.

Si utiliza ConnectionConsumer, IBM WebSphere MQ Resource Adapter o el proveedor de mensajería WebSphere Application Server IBM WebSphere MQ , es posible que necesite autorización adicional.

Las colas que debe leer ConnectionConsumer deben tener las autorizaciones `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 de proveedor de mensajería de WebSphere MQ para realizar la mensajería de publicación/suscripción, la aplicación utiliza la funcionalidad de publicación/suscripción integrada proporcionada por 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 WebSphere MQ classes for JMS

Una aplicación WebSphere MQ para JMS puede conectarse a un gestor de colas en modalidad de cliente o de enlaces. En modalidad de cliente, WebSphere MQ classes for JMS se conecta al gestor de colas a través de TCP/IP. En modalidad de enlaces, WebSphere MQ classes for JMS se conecta directamente al gestor de colas utilizando JNI (Java Native Interface).

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 puede conectarse a un gestor de colas sólo 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 soportada actual o anterior de WebSphere 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 WebSphere 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 conectarse a un gestor de colas en modalidad de cliente, una aplicación WebSphere MQ classes for JMS puede ejecutarse en el mismo sistema en el que se ejecuta el gestor de colas, o en un sistema diferente. En cada caso, WebSphere MQ classes for JMS se conecta al gestor de colas a través de TCP/IP.

Modalidad de enlaces

Para conectarse a un gestor de colas en modalidad de enlaces, una aplicación WebSphere MQ classes for JMS debe ejecutarse en el mismo sistema en el que se ejecuta el gestor de colas.

Las clases WebSphere MQ para JMS se conectan directamente al gestor de colas utilizando JNI (Java Native Interface). Para utilizar el transporte de enlaces, las clases de WebSphere MQ para JMS deben ejecutarse en un entorno que tenga acceso a las bibliotecas de la interfaz nativa Java de WebSphere MQ ; consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 739 para obtener más información.

Las clases de WebSphere MQ para JMS dan soporte a los valores siguientes para *ConnectOption* :

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Para cambiar las opciones de conexión utilizadas por las clases WebSphere MQ para JMS, modifique la propiedad CONNOPTde 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 213

Para utilizar el transporte de enlaces, el entorno de ejecución de Java que se utiliza debe dar soporte al CCSID (Coded Character Set Identifier) del gestor de colas al que se conectan las clases de WebSphere MQ para JMS .

Los detalles sobre cómo determinar qué CCSID están soportados por un entorno de ejecución Java se pueden encontrar en WebSphere MQ FDC con el ID de analizador 21 generado al utilizar las clases WebSphere MQ V7 para Java o WebSphere MQ V7 clases para JMS .

Enlaces y, a continuación, modalidad de cliente

Éste es el valor predeterminado. Al conectarse a un gestor de colas en esta modalidad, una aplicación WebSphere MQ classes for JMS intentará conectarse en modalidad de enlaces, lo que requiere que el gestor de colas resida en la misma máquina que la aplicación. Si la conexión no es satisfactoria, la aplicación intentará conectarse en modalidad de cliente, permitiendo que el gestor de colas resida localmente en la misma máquina que la aplicación o de forma remota.

Configuración del gestor de colas para que las clases de WebSphere MQ para aplicaciones JMS puedan conectarse en modalidad de cliente

Para configurar el gestor de colas para que las clases de WebSphere MQ para aplicaciones JMS se puedan conectar en modalidad de cliente, debe crear una definición de canal de conexión de servidor e iniciar un escucha.

En z/OS, la característica Client Attachment debe estar instalada.

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)
```

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.

También puede crear una definición de canal de conexión de servidor utilizando IBM WebSphere MQ Explorer, que se ejecuta en Linux y Windows, o los paneles de operaciones y control en z/OS.

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.

En todas las plataformas, puede utilizar el mandato MQSC START LISTENER para iniciar un escucha, pero, excepto en z/OS, primero debe crear un objeto de escucha utilizando el mandato MQSC DEFINE LISTENER. Vea el ejemplo siguiente:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

En sistemas UNIX, Linuxy 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.

También puede iniciar un escucha utilizando WebSphere MQ Explorer, que se ejecuta en Linux y Windows, o los paneles de operaciones y control en z/OS.

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.

La prueba de verificación de instalación punto a punto para las clases WebSphere MQ para JMS

Se proporciona un programa de prueba de verificación de instalación punto a punto (IVT) con clases WebSphere MQ para JMS. 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 "Tipos de objeto JMS" en la página 958.

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.

Se suministra un script para ejecutar el programa IVT. El script se denomina IVTRun en sistemas UNIX and Linux y IVTRun.bat en Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de WebSphere MQ para JMS.

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 "Preparación y ejecución de los programas de ejemplo" en la página 112. 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]
```

No se proporciona ningún script equivalente en sistemas z/OS , pero puede ejecutar la IVT en modalidad de enlaces invocando la clase Java directamente, utilizando el mandato siguiente:

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

La vía de acceso de clases debe contener com.ibm.mqjms.jar.

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. Para obtener más información sobre este parámetro, incluidos sus valores válidos, consulte la descripción de la propiedad PROVIDERVERSION en Propiedades de objetos IBM WebSphere 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 activado. De forma predeterminada, el rastreo está desactivado.

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: WebSphere MQ Client for Java
    JMSXDeliveryCount: 1
    JMS_IBM_PutDate: 20070815
    JMS_IBM_PutTime: 09310400
    JMS_IBM_Format: MQSTR
    JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Prueba de verificación de la instalación punto a punto 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 JMS de WebSphere MQ para que pueda utilizar el servicio de directorio para almacenar objetos administrados. Para obtener más información sobre estos requisitos previos, consulte “Requisitos previos para WebSphere MQ classes for JMS” en la página 731. Para obtener información sobre cómo configurar la herramienta de administración JMS de WebSphere MQ , consulte “Configuración de la herramienta de administración JMS” en la página 954.

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 sistemas UNIX and Linux y

IVTSetup.bat en Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de WebSphere MQ classes for JMS. Para ejecutar el script, entre el mandato siguiente:

```
IVTSetup
```

El script invoca la herramienta de administración JMS de WebSphere MQ 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 modalidad de cliente, o se conecte a un gestor de colas que no sea el gestor de colas predeterminado, debe utilizar la herramienta de administración JMS de WebSphere MQ o WebSphere MQ Explorer para cambiar las propiedades adecuadas del objeto de fábrica MQQueueConnection. Para obtener información sobre cómo utilizar la herramienta de administración JMS de WebSphere MQ , consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953. Para obtener información sobre cómo utilizar WebSphere MQ Explorer, consulte la ayuda proporcionada con WebSphere 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 activado. De forma predeterminada, el rastreo está desactivado.

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 sistemas UNIX and Linux y IVTTidy.bat en Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de WebSphere 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 "Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS" en la página 737.
- 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 "Preparación y ejecución de los programas de ejemplo" en la página 112 para obtener los detalles.

La prueba de verificación de la instalación de publicación/suscripción para las clases WebSphere MQ para JMS

Se proporciona un programa de prueba de verificación de instalación (IVT) de publicación/suscripción con clases WebSphere MQ para 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 "Tipos de objeto JMS" en la página 958.

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 sistemas UNIX and Linux y PSIVTRun.bat en Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de las clases de WebSphere MQ para 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 "Preparación y ejecución de los programas de ejemplo" en la página 112 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 sólo es relevante si el parámetro -v especifica un número de versión de gestor de colas menor que 7 y está utilizando WebSphere Event Broker o WebSphere Message Broker como intermediario de publicación/suscripción.

-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. Para obtener más información sobre este parámetro, incluidos sus valores válidos, consulte la descripción de la propiedad PROVIDERVERSION en Propiedades de objetos IBM WebSphere 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 activado. De forma predeterminada, el rastreo está desactivado.

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
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 JMS de WebSphere MQ para que pueda utilizar el servicio de directorio para almacenar objetos administrados. Para obtener más información sobre estos requisitos previos, consulte “Requisitos previos para WebSphere MQ classes for JMS” en la página 731. Para obtener información sobre cómo configurar la herramienta de administración JMS de WebSphere MQ , consulte “Configuración de la herramienta de administración JMS” en la página 954.

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 sistemas UNIX and Linux y IVTSetup.bat en

Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de WebSphere MQ classes for JMS. Para ejecutar el script, entre el mandato siguiente:

```
IVTSetup
```

El script invoca la herramienta de administración JMS de WebSphere MQ 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 modalidad de cliente, conéctese a un gestor de colas que no sea el gestor de colas predeterminado, o utilice WebSphere Event Broker o WebSphere Message Broker en lugar de la función de publicación/suscripción incorporada, debe utilizar la herramienta de administración JMS de WebSphere MQ o WebSphere MQ Explorer para cambiar las propiedades adecuadas del objeto de fábrica MQTopicConnection. Para obtener información sobre cómo utilizar la herramienta de administración JMS de WebSphere MQ , consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953. Para obtener información sobre cómo utilizar WebSphere MQ Explorer, consulte la ayuda proporcionada con WebSphere 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 activado. De forma predeterminada, el rastreo está desactivado.

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 sistemas UNIX and Linux y IVTTidy.bat en Windows, y se encuentra en el subdirectorio `bin` del directorio de instalación de WebSphere 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 "Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS" en la página 737.
- 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 "Preparación y ejecución de los programas de ejemplo" en la página 112.

El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ

El programa IVT se proporciona como un archivo EAR. Para utilizar el programa, debe desplegarlo y definir algunos objetos como recursos JCA.

El programa de prueba de verificación de instalación (IVT) se proporciona como un archivo EAR (Enterprise Archive) denominado wmq.jmsra.ivt.ear. Este archivo se instala con WebSphere MQ classes for JMS en el mismo directorio que el archivo RAR del adaptador de recursos de WebSphere MQ , wmq.jmsra.rar. Para obtener información sobre dónde se instalan estos archivos, consulte "Instalación del adaptador de recursos WebSphere MQ" en la página 749.

Debe desplegar el programa IVT en el servidor de aplicaciones. El programa IVT incluye un servlet y un MDB que prueba que un mensaje se puede enviar a, y recibir de, una cola de WebSphere MQ .

Opcionalmente, puede utilizar el programa IVT para verificar que el adaptador de recursos de WebSphere MQ se ha configurado correctamente para dar soporte a transacciones distribuidas.

Para poder 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 a un espacio de nombres JNDI. Puede elegir las propiedades de los objetos, pero el siguiente conjunto de propiedades es un ejemplo sencillo:

objeto ConnectionFactory

```
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1414
queueManager:         ExampleQM
transportType:        CLIENT
```

Objeto de cola

```
baseQueueManagerName:  ExampleQM
baseQueueName:         TEST.QUEUE
```

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.

Después de haber desplegado el programa IVT y de que el servidor de aplicaciones haya creado los objetos JMS y los haya enlazado al espacio de nombres JNDI, puede iniciar el programa IVT especificando un URL con el formato siguiente en el navegador web:

```
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/
```

La Figura 122 en la página 800 muestra la página inicial del programa IVT.

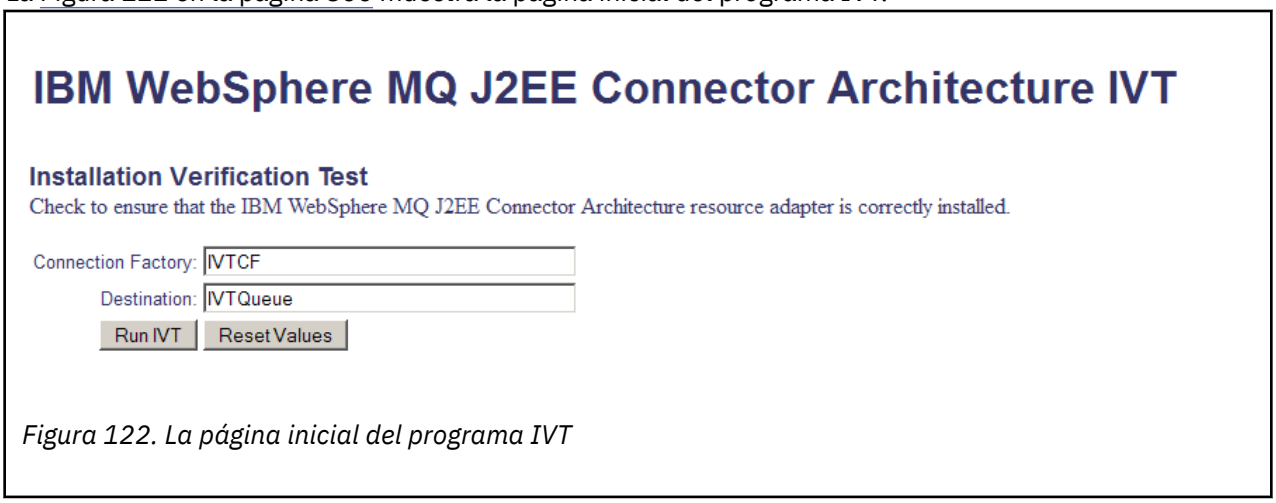


Figura 122. La página inicial del programa IVT

Para ejecutar la prueba, pulse **Ejecutar IVT**.Figura 123 en la página 801 muestra la página que se visualiza si la IVT es satisfactoria.

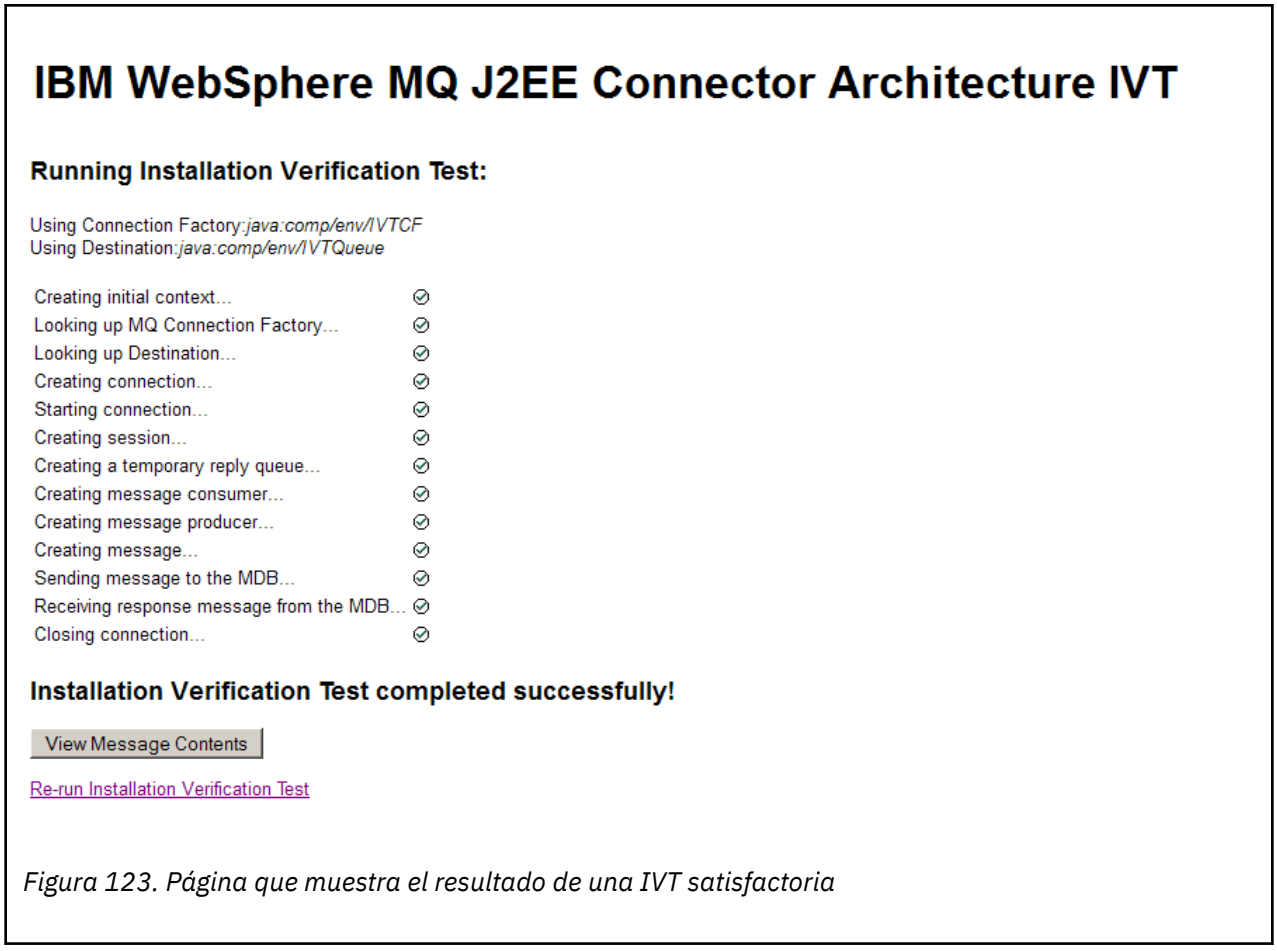


Figura 123. Página que muestra el resultado de una IVT satisfactoria

Si la prueba IVT falla, se visualiza una página como la que se muestra en la Figura 124 en la página 802. Para obtener información adicional sobre la causa del error, pulse **Ver rastreo de pila**.

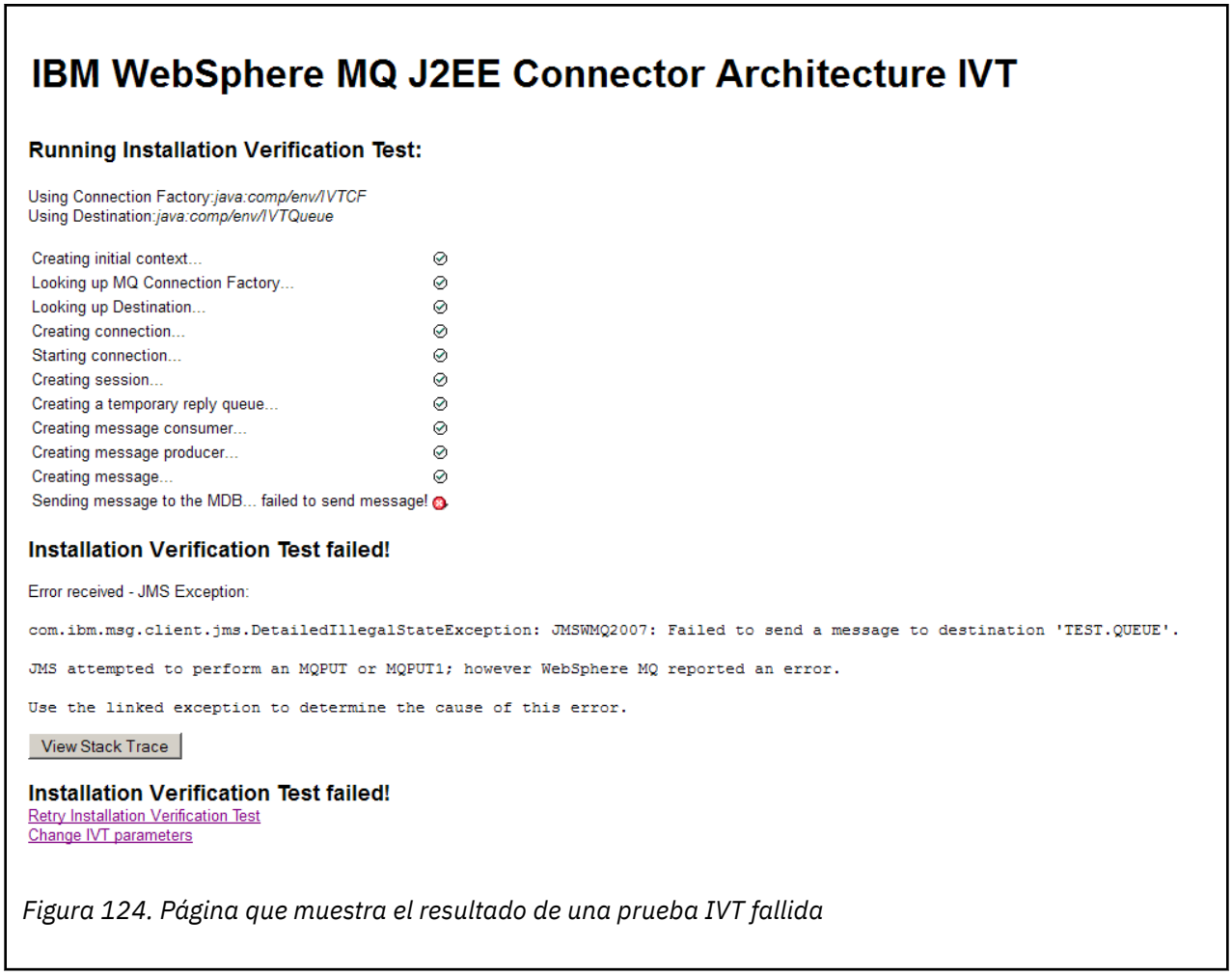


Figura 124. Página que muestra el resultado de una prueba IVT fallida

Para obtener instrucciones detalladas e información sobre los scripts de programa de utilidad proporcionados para desplegar la aplicación IVT en los servidores de aplicaciones JBoss y WAS CE, consulte:

Tareas relacionadas

“Instalación y prueba del adaptador de recursos MQ en WAS CE” en la página 802
Instalación del adaptador de recursos de IBM WebSphere MQ y ejecución de la aplicación de prueba de verificación de instalación (IVT) en WebSphere Application Server CE.

“Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6” en la página 805
Después de instalar el adaptador de recursos de IBM WebSphere MQ en JBoss AS 5.1 o 6, puede probar la instalación del adaptador de recursos instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Instalación y prueba del adaptador de recursos MQ en WAS CE

Instalación del adaptador de recursos de IBM WebSphere MQ y ejecución de la aplicación de prueba de verificación de instalación (IVT) en WebSphere Application Server CE.

Antes de empezar

Esta tarea presupone que tiene un servidor WebSphere Application Server CE 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 WebSphere MQ en el sistema local y que está familiarizado con las tareas de administración estándar.

Si está utilizando el adaptador de recursos para conectarse a un cliente IBM WebSphere MQ y necesita realizar transacciones XA distribuidas, debe seguir los pasos adicionales marcados como **Sólo XA de cliente**.

1. Cree un gestor de colas denominado `ExampleQMy` establézcalo tal como se describe en "Preparación y ejecución de los programas de ejemplo" en la página 112 , teniendo en cuenta que el escucha se debe iniciar en el puerto 1414, el canal que se va a utilizar se denomina `SYSTEM.DEF.SVRCONN` y la cola utilizada por la aplicación IVT se denomina `TEST.QUEUE`. A la cola modelo `SYSTEM.DEFAULT.MODEL.QUEUE` también se le tendrá que otorgar la autorización DSP y PUT para que esta aplicación pueda crear una cola de respuestas temporal. Si desea utilizar un gestor de colas diferente, detalles de conexión diferentes o una cola diferente, consulte "Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado" en la página 804.
2. Obtenga el archivo de adaptador de recursos (`wmq.jmsra.rar`), la aplicación IVT (`wmq.jmsra.ivt.ear`) y `WAS_CE_jmsra_deployment_plan.xml` y `WAS_CE_jmsra_ivt_deployment_plan.xml deployment plan files`. Para obtener detalles sobre la ubicación de estos archivos, consulte "Instalación del adaptador de recursos WebSphere MQ" en la página 749.

Para obtener una descripción de los enlaces y las conexiones en modalidad de cliente, consulte "Modalidades de conexión para WebSphere MQ classes for JMS" en la página 789.

Si desea utilizar una cola, un gestor de colas, un puerto, un host, un canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, consulte "Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado" en la página 804.

Procedimiento

1. **Sólo XA de cliente**: edite la copia del archivo `WAS_CE_jmsra_deployment_plan.xml` .
   a) Busque la definición de conexión jms/ivt/IVTCF y modifíquela para que la fábrica de conexiones esté habilitada para transacciones XA.
      i) Comente la sección NonXA :

      ```
      <conn:xa-transaction>
      ```

      ii) Elimine el comentario de la sección de configuración XA:

      ```
      <conn:xa-transaction>
          <conn:transaction-caching/>
      </conn:xa-transaction>
      ```

   b) Guarde los cambios.
2. Opcional: **Sólo XA de cliente**: modifique el descriptor de ensamblaje del MDB para que requiera transacciones. Esto fuerza al MDB en la IVT a participar en una transacción XA, aunque la aplicación IVT sigue funcionando sin esta modificación.
   a) Abra el archivo `wmq.jmsra.ivt.ear`.
   b) Abra el archivo `WMQ_IVT_MDB.jar` dentro de él.
   c) Editar `META-INF/ejb-jar.xml`.
      i) Comente o suprima la línea dentro del descriptor de ensamblaje:

      ```
      <trans-attribute>NotSupported</trans-attribute>
      ```

      ii) Elimine el comentario de la línea dentro del descriptor de ensamblaje:

      ```
      <trans-attribute>Required</trans-attribute>
      ```

      iii) Guarde los cambios y actualice el archivo dentro del archivo `WMQ_IVT_MDB.jar` .
      iv) Actualice el archivo wmq.jmsra.ivt.ear con el archivo `WMQ_IVT_MDB.jar` modificado.

3. Despliegue el adaptador de recursos en el servidor utilizando el archivo de plan de despliegue modificado.
   a) Para hacerlo en la línea de mandatos, escriba el siguiente mandato WAS CE:

   ```
   deploy -u system -p manager deploy wmq.jmsra.rar WAS_CE_jmsra_deployment_plan.xml
   ```

   b) Utilizando la interfaz de administración web, vaya a **Aplicaciones > Desplegador:**
      i) Establezca el archivo de archivado en el archivo wmq.jmsra.rar .
      ii) Establezca el plan para que sea el archivo WAS_CE_jmsra_deployment_plan.xml .
      iii) Asegúrese de que se ha seleccionado 'Iniciar aplicación después de la instalación'.
      iv) Pulse **Instalar**.
4. Despliegue la aplicación IVT en el servidor utilizando el plan de despliegue proporcionado.
   a) En la línea de mandatos, esto se puede realizar utilizando el siguiente mandato WAS CE:

   ```
   deploy -u system -p manager deploy wmq.jmsra.ivt.ear WAS_CE_jmsra_ivt_deployment_plan.xml
   ```

   b) Utilizando la interfaz de administración web, vaya a **Aplicaciones > Desplegador:**
      i) Establezca el **Archivador** para que sea el archivo `wmq.jmsra.ivt.ear` .
      ii) Establezca el **Plan** para que sea el archivo `WAS_CE_jmsra_ivt_deployment_plan.xml` .
      iii) Asegúrese de que **Iniciar aplicación tras instalación** está seleccionado.
      iv) Pulse **Instalar**.
5. Ejecute la aplicación IVT. Para obtener más información, consulte "El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ" en la página 799. Para WAS CE, el URL predeterminado es `http://localhost:8080/WMQ_IVT/`.

Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado

Si desea utilizar una cola, un gestor de colas, un puerto, un host, un canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, debe modificar la aplicación IVT y los scripts asociados en WebSphere Application Server CE antes de desplegar el adaptador de recursos o la aplicación IVT.

Acerca de esta tarea

Si desea realizar el despliegue en una configuración distinta de la que se especifica en "Instalación y prueba del adaptador de recursos MQ en WAS CE" en la página 802, es decir, si desea utilizar una cola, un gestor de colas, un puerto, un host, un canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, realice los pasos siguientes antes de desplegar el adaptador de recursos o la aplicación IVT.

Procedimiento

1. Si desea especificar un gestor de colas y una cola diferentes para utilizar para la aplicación IVT, establezca valores para el gestor de colas y la cola en `WAS_CE_jmsra_deployment_plan.xml`, para obtener detalles, consulte "Establecimiento de valores para el gestor de colas y la cola" en la página 805.
2. Si desea especificar un gestor de colas y una cola diferentes en la configuración para el bean controlado por mensajes (MDB), establezca valores para el gestor de colas y la cola que está utilizando en `WAS_CE_jmsra_ivt_deployment_plan.xml`, para obtener detalles, consulte "Establecimiento de valores para la configuración de MDB" en la página 805.
3. Si está configurando el adaptador de recursos para conectarse a IBM WebSphere MQ en modalidad de enlaces, asegúrese de que las bibliotecas JNI estén en la vía de acceso del sistema o en la vía de acceso de WAS CE. Para obtener más información, consulte "Instalación y prueba del adaptador de recursos MQ en WAS CE" en la página 802.
4. Si ya ha desplegado el adaptador de recursos, puede volver a desplegarlo con el plan de despliegue modificado para cambiar los valores utilizando el mandato siguiente:

```
deploy --user system --password manager redeploy wmq.jmsra.rar
WAS_CE_jmsra_deployment_plan.xml
```

Qué hacer a continuación

Continúe desplegando el adaptador de recursos tal como se describe en "Instalación y prueba del adaptador de recursos MQ en WAS CE" en la página 802.

Establecimiento de valores para el gestor de colas y la cola

Explica cómo establecer valores para el gestor de colas y la cola que está utilizando en `WAS_CE_jmsra_deployment_plan.xml`.

Procedimiento

En `WAS_CE_jmsra_deployment_plan.xml`, establezca valores para el gestor de colas y la cola que está utilizando para la aplicación IVT.

Para la definición de conexión jms/ivt/IVTCF:

1. Establezca el valor del elemento queueManager para que sea el nombre del gestor de colas.
2. Si está utilizando una conexión de cliente, establezca el valor de los diversos elementos de conexión de cliente para que sea adecuado para una conexión con el gestor de colas.
3. Si está utilizando una conexión de enlaces:
   a. Establezca el valor del elemento transportType en BINDINGS.
   b. Comente o suprima los diversos elementos de conexión de cliente.
4. Para el destino de mensajes jms/ivt/IVTQueue, establezca el valor del elemento de nombre baseQueuepara que sea el nombre de la cola que ha creado para la aplicación IVT
5. Guarde los cambios.

Establecimiento de valores para la configuración de MDB

Explica cómo establecer valores para la configuración de MDB en `WAS_CE_jmsra_deployment_plan.xml`.

Procedimiento

En `WAS_CE_jmsra_ivt_deployment_plan.xml` , establezca valores para el gestor de colas y la cola que está utilizando en la configuración para el MDB.

Para el bean controlado por mensajes WMQ_IVT_MDB:

1. Establezca el valor del elemento queueManager para que sea el nombre del gestor de colas.
2. Si está utilizando una conexión de cliente, establezca el valor de los diversos elementos de conexión de cliente para que sea adecuado para una conexión con el gestor de colas.
3. Si está utilizando una conexión de enlaces:
   a. Establezca el valor del elemento transportType en BINDINGS.
   b. Comente o suprima los diversos elementos de conexión de cliente.
4. Guarde los cambios.

Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6

Después de instalar el adaptador de recursos de IBM WebSphere MQ en JBoss AS 5.1 o 6, puede probar la instalación del adaptador de recursos instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Antes de empezar

Importante: Estas instrucciones son para JBoss AS 5.1 y 6, no son válidas para JBoss AS 7.

Para obtener información sobre la instalación del adaptador de recursos en JBoss EAP 6.3, consulte "Instalación y prueba del adaptador de recursos en JBoss EAP 6.3" en la página 808.

Esta tarea presupone que tiene un servidor JBoss en ejecución y que está familiarizado con las tareas de administración estándar para el mismo. Esta tarea presupone que tiene una instalación de IBM WebSphere MQ en el sistema local y que está familiarizado con las tareas de administración estándar.

Si está utilizando el adaptador de recursos para conectarse a un cliente IBM WebSphere MQ y necesita realizar transacciones XA distribuidas, debe seguir los pasos adicionales marcados **Sólo XA de cliente**. Para obtener una descripción de los enlaces y las conexiones en modalidad de cliente, consulte "Modalidades de conexión para WebSphere MQ classes for JMS" en la página 789.

Procedimiento

1. Cree un gestor de colas denominado ExampleQMy establézcalo tal como se describe en "Preparación y ejecución de los programas de ejemplo" en la página 112.

   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.

   Si desea utilizar un gestor de colas diferente, detalles de conexión diferentes o una cola diferente, consulte "Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado" en la página 804.
2. Obtenga el archivo de adaptador de recursos (`wmq.jmsra.rar`), la aplicación IVT (`wmq.jmsra.ivt.ear`) y el archivo `jboss-jmsra-ds.xml` .

   Para ver la ubicación de estos archivos, consulte "Instalación del adaptador de recursos WebSphere MQ" en la página 749.
3. **Sólo XA de cliente**: Edite el archivo `jboss-jmsra-ds.xml` para habilitar las transacciones XA en la fábrica de conexiones.
   a) Comente o suprima la línea dentro de la definición de fábrica de conexiones `<local-transaction/>`.
   b) Elimine el comentario de la línea dentro de la definición de fábrica de conexiones `<xa-transaction/>`.
   c) Guarde los cambios.
4. **Sólo XA de cliente**: (opcional) modifique el descriptor de ensamblaje del MDB para que requiera transacciones. Esto fuerza al MDB en la IVT a participar en una transacción XA, aunque la aplicación IVT sigue funcionando sin esta modificación.
   a) Abra el archivo `wmq.jmsra.ivt.ear`.
   b) Abra el archivo `WMQ_IVT_MDB.jar` dentro de él.
   c) Edite `META-INF/ejb-jar.xml`:
      i) Comente o suprima la línea dentro del descriptor de ensamblaje:

      ```
      <trans-attribute>NotSupported</trans-attribute>
      ```

      ii) Elimine el comentario de la línea dentro del descriptor de ensamblaje:

      ```
      <trans-attribute>Required</trans-attribute>
      ```

      iii) Guarde los cambios y actualice el archivo dentro del archivo `WMQ_IVT_MDB.jar` .

iv) Actualice el archivo `wmq.jmsra.ivt.ear` con el `WMQ_IVT_MDB.jar`modificado.

5. Despliegue el adaptador de recursos en el servidor copiando el archivo `wmq.jmsra.rar` en el directorio `jboss/server/default/deploy`.
6. Cree los recursos JMS necesarios para la aplicación IVT copiando el archivo `jboss-jmsra-ds.xml` en el directorio `jboss/server/default/deploy`.
7. Despliegue la aplicación IVT copiando el archivo `wmq.jmsra.ivt.ear` en el directorio `jboss/server/default/deploy`.
8. Ejecute la aplicación IVT. Para obtener más información, consulte "El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ" en la página 799. Para JBoss, el URL predeterminado es `http://localhost:8080/WMQ_IVT/`.

Despliegue de la aplicación IVT en JBoss con un entorno IBM WebSphere MQ personalizado

Al instalar el adaptador de recursos de IBM WebSphere MQ en JBoss, si desea utilizar una cola, gestor de colas, puerto, host, canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, primero debe modificar la aplicación IVT y los scripts asociados en JBoss antes de desplegar el adaptador de recursos o la aplicación IVT.

Acerca de esta tarea

Importante: Estas instrucciones sólo son aplicables para Java EE Versiones 6 y 5, no para Java EE Versión 7. Por lo tanto, el uso de estas instrucciones para JBoss Versión 8 (WildFly) no está soportado.

Si desea desplegar en una configuración distinta de la especificada en "Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6" en la página 805, es decir, si desea utilizar un gestor de colas, cola, puerto, host, canal o utilizar la modalidad de enlaces en lugar de la modalidad de cliente, realice los pasos siguientes antes de desplegar el adaptador de recursos o la aplicación IVT.

Procedimiento

1. Si desea especificar un gestor de colas y una cola diferentes para utilizar para la aplicación IVT, establezca valores para el gestor de colas y la cola.
   a) Para la definición de conexión jms/ivt/IVTCF:
      i) Establezca el valor de la propiedad de configuración queueManager para que sea el nombre del gestor de colas.
      ii) Si está utilizando una conexión de cliente, establezca el valor de los diversos elementos de conexión de cliente para que sea adecuado para una conexión con el gestor de colas.
      iii) Si está utilizando una conexión de enlaces, establezca el valor del elemento transportType en BINDINGS y, a continuación, comente o suprima los distintos elementos de conexión de cliente.
   b) Para el mbean jms/ivt/IVTQueue, establezca el valor del elemento de nombre baseQueuepara que sea el nombre de la cola que ha creado para la aplicación IVT.
   c) Guarde los cambios.
2. Si desea especificar un gestor de colas y una cola diferentes en la configuración para el bean controlado por mensajes (MDB), modifique la configuración del MDB para conectarse al gestor de colas y a la cola.
   a) Abra el archivo `wmq.jmsra.ivt.ear`.
   b) Abra el `WMQ_IVT_MDB.jar` dentro del mismo.
   c) Edite `META-INF/ejb-jar.xml`:
      i) Establezca el valor de la propiedad activation-config-property queueManager para que sea el nombre del gestor de colas.

ii) Si está utilizando una conexión de cliente, establezca el valor de las distintas propiedades de activación de conexión de cliente para que sea adecuado para una conexión con el gestor de colas.

iii) Si está utilizando una conexión de enlaces, establezca el valor de activation-config-property transportType en BINDINGS y, a continuación, comente o suprima los distintos elementos de conexión de cliente.

d) Guarde los cambios y actualice el archivo dentro del archivo `WMQ_IVT_MDB.jar` .

e) Actualice el archivo `wmq.jmsra.ivt.ear` con el `WMQ_IVT_MDB.jar`modificado.

3. Si está configurando el adaptador de recursos para conectarse a IBM WebSphere MQ en modalidad de enlaces, asegúrese de que las bibliotecas JNI estén en la vía de acceso del sistema o en la vía de acceso de JBoss. Para obtener más información, consulte "Configuración de las bibliotecas JNI (Java Native Interface)" en la página 739.

Qué hacer a continuación

Continúe desplegando el adaptador de recursos tal como se describe en "Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6" en la página 805.

.

Instalación y prueba del adaptador de recursos en JBoss EAP 6.3

Después de instalar el adaptador de recursos de IBM WebSphere MQ en JBoss Enterprise Application Platform (EAP) 6.3, ya sea en un servidor autónomo o en un servidor que se ejecuta en un dominio gestionado, puede probar la instalación del adaptador de recursos instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Acerca de esta tarea

Importante: Estas instrucciones son únicamente para JBoss EAP 6.3 . Para obtener información sobre la instalación del adaptador de recursos en JBoss AS 5.1 y 6, consulte "Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6" en la página 805.

Esta tarea presupone que tiene un servidor JBoss 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 WebSphere MQ en el sistema local y que está familiarizado con la administración estándar.

Procedimiento

1. Cree un gestor de colas denominado ExampleQMy establézcalo tal como se describe en "Preparación y ejecución de los programas de ejemplo" en la página 112.

   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.

   Si desea utilizar un gestor de colas diferente, detalles de conexión diferentes o una cola diferente, consulte "Despliegue de la aplicación IVT en WAS CE con un entorno MQ personalizado" en la página 804.
2. Obtenga el archivo de adaptador de recursos (`wmq.jmsra.rar`) y la aplicación IVT (`wmq.jmsra.ivt.ear`).

   Para ver la ubicación de estos archivos, consulte "Instalación del adaptador de recursos WebSphere MQ" en la página 749.

3. Instale el adaptador de recursos y, a continuación, pruebe la instalación ejecutando la aplicación de prueba de verificación de instalación (IVT):
   - Si está instalando el adaptador de recursos en un servidor autónomo, consulte "Instalación y prueba en un servidor autónomo" en la página 809.
   - Si está instalando el adaptador de recursos en un servidor que se ejecuta en un dominio gestionado, consulte "Instalación y prueba en un servidor que se ejecuta en un dominio gestionado" en la página 810.

Instalación y prueba en un servidor autónomo

Después de instalar el adaptador de recursos de IBM WebSphere MQ en JBoss EAP 6.3 en un servidor autónomo, puede probar la instalación del adaptador de recursos instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Acerca de esta tarea

La información de esta tarea es para instalar y probar el adaptador de recursos en un servidor autónomo. Si está instalando el adaptador de recursos en un servidor que se ejecuta en un dominio gestionado, consulte "Instalación y prueba en un servidor que se ejecuta en un dominio gestionado" en la página 810.

Importante: Estas instrucciones son únicamente para JBoss EAP 6.3 . Para obtener información sobre la instalación del adaptador de recursos en JBoss AS 5.1 y 6, consulte "Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6" en la página 805.

Procedimiento

1. Despliegue el adaptador de recursos en el servidor copiando el archivo `wmq.jmsra.rar` en el directorio <*EAP_HOME*>`/standalone/deployments`.
2. Cree los recursos JMS necesarios para la aplicación IVT añadiendo las entradas siguientes a la sección `<resource-adapters>` del archivo <*EAP_HOME*>`/standalone/configuration/standalone-full.xml` :

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
     <connection-definition
          class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
          jndi-name="java:jboss/jms/ivt/IVTCF"
          enabled="true"
          use-java-context="true"
          pool-name="IVTCF">
       <config-property name="port">
           1414
       </config-property>
       <config-property name="hostName">
           localhost
       </config-property>
       <config-property name="channel">
           SYSTEM.DEF.SVRCONN
       </config-property>
       <config-property name="transportType">
           CLIENT
       </config-property>
       <config-property name="queueManager">
           ExampleQM
       </config-property>
     </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
          TEST.QUEUE
      </config-property>
```

```
    </admin-object>
  </admin-objects>
</resource-adapter>
```

3. Añada la siguiente información a los parámetros de inicio del servidor de aplicaciones:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Despliegue la aplicación IVT copiando el archivo `wmq.jmsra.ivt.ear` en el directorio <*EAP_HOME*>/`standalone/deployments`.
5. Inicie el servidor de aplicaciones.
6. Ejecute la aplicación IVT.

   Para obtener más información, consulte "El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ" en la página 799. Para JBoss, el URL predeterminado es `http://localhost:8080/WMQ_IVT/`.

   Nota: Los nombres JNDI utilizados para los recursos JMS necesarios para ejecutar la aplicación IVT son los siguientes:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   Cuando inicie la aplicación IVT utilizando el URL especificado anteriormente, especifique los nombres JNDI de estos recursos en sus respectivos campos y, a continuación, ejecute la aplicación pulsando **Ejecutar IVT**.

Instalación y prueba en un servidor que se ejecuta en un dominio gestionado

Después de instalar el adaptador de recursos de IBM WebSphere MQ en JBoss EAP 6.3 en un servidor que se ejecuta en un dominio gestionado, puede probar la instalación del adaptador de recursos instalando y ejecutando la aplicación de prueba de verificación de instalación (IVT).

Acerca de esta tarea

La información de esta tarea es para instalar y probar el adaptador de recursos en un servidor que se ejecuta en un dominio gestionado. Si está instalando el adaptador de recursos en un servidor autónomo, consulte "Instalación y prueba en un servidor autónomo" en la página 809.

Importante: Estas instrucciones son únicamente para JBoss EAP 6.3 . Para obtener información sobre la instalación del adaptador de recursos en JBoss AS 5.1 y 6, consulte "Instalación y prueba del adaptador de recursos en JBoss AS 5.1 y 6" en la página 805.

Procedimiento

1. Despliegue el adaptador de recursos en el servidor utilizando la consola de gestión de JBoss o la CLI de gestión.
2. Cree los recursos JMS necesarios para la aplicación IVT añadiendo las entradas siguientes a la sección `<resource-adapters>` de `<EAP_HOME>/domain/configuration/domain.xml file`:

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
    <connection-definition
         class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
         jndi-name="java:jboss/jms/ivt/IVTCF"
         enabled="true"
         use-java-context="true"
         pool-name="IVTCF">
      <config-property name="port">
          1414
```

```
      </config-property>
      <config-property name="hostName">
          localhost
      </config-property>
      <config-property name="channel">
          SYSTEM.DEF.SVRCONN
      </config-property>
      <config-property name="transportType">
          CLIENT
      </config-property>
      <config-property name="queueManager">
          ExampleQM
      </config-property>
    </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
          TEST.QUEUE
      </config-property>
    </admin-object>
  </admin-objects>
</resource-adapter>
```

3. Añada la siguiente información a los parámetros de inicio del servidor de aplicaciones:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Detenga y reinicie el servidor de aplicaciones.
5. Despliegue la aplicación IVT utilizando la consola de gestión de JBoss o la CLI de gestión.
6. Ejecute la aplicación IVT.

   Para obtener más información, consulte "El programa de prueba de verificación de instalación para el adaptador de recursos WebSphere MQ" en la página 799. Para JBoss, el URL predeterminado es `http://localhost:8080/WMQ_IVT/`.

   Nota: Los nombres JNDI utilizados para los recursos JMS necesarios para ejecutar la aplicación IVT son los siguientes:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   Especifique los nombres JNDI de estos recursos en sus respectivos campos cuando inicie la aplicación IVT utilizando el URL especificado anteriormente y, a continuación, ejecute la aplicación pulsando **Ejecutar IVT**.

Scripts proporcionados con WebSphere MQ classes for JMS

Se proporcionan varios scripts para ayudar con las tareas comunes que se deben realizar cuando se utilizan clases WebSphere MQ para JMS.

En la Tabla 102 en la página 811 se incluye la lista de todos los scripts y sus utilizaciones. Los scripts se encuentran en el subdirectorio `bin` del directorio de instalación de WebSphere MQ classes for JMS.

Tabla 102. Scripts proporcionados con WebSphere MQ classes for JMS

Programa de utilidad	**Uso**
Limpieza[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.

Tabla 102. Scripts proporcionados con WebSphere MQ classes for JMS (continuación)	
Programa de utilidad	**Uso**
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 "La prueba de verificación de instalación punto a punto para las clases WebSphere MQ para JMS" en la página 791.
JMSAdmin[1]	Ejecuta la herramienta de administración JMS de WebSphere MQ , tal como se describe en "Invocación de la herramienta de administración de IBM WebSphere MQ classes for JMS" en la página 953.
JMSAdmin.config	El archivo de configuración para la herramienta de administración JMS de WebSphere MQ , tal como se describe en "Configuración de la herramienta de administración JMS" en la página 954.
PSIVTRun[1]	Ejecuta el programa de prueba de verificación de la instalación de publicación/suscripción, tal como se describe en el apartado "La prueba de verificación de la instalación de publicación/suscripción para las clases WebSphere MQ para JMS" en la página 795.
PSReportDump.class	Esta clase se conserva para mantener la compatibilidad con releases anteriores, pero no realiza ninguna función.
setjmsenv	Establece las variables de entorno para ejecutar una aplicación WebSphere MQ para JMS en una máquina virtual Java (JVM) de 32 bits en sistemas UNIX and Linux , tal como se describe en "Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS" en la página 737.
setjmsenv64	Establece las variables de entorno para ejecutar una aplicación WebSphere MQ para JMS en una JVM de 64 bits en sistemas UNIX and Linux , tal como se describe en "Variables de entorno utilizadas por las clases IBM WebSphere MQ para JMS" en la página 737.
Nota: 1. En Windows, el nombre de archivo tiene la extensión .bat.	

Soporte para OSGi

OSGi proporciona una infraestructura que da soporte al despliegue de aplicaciones como paquetes. Se proporcionan nueve paquetes OSGi como parte de IBM WebSphere MQ classes for JMS.

OSGi proporciona una infraestructura Java con fines generales, segura y gestionada, que soporta el despliegue de aplicaciones en forma 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.

IBM WebSphere MQ classes for JMS. incluye los siguientes paquetes OSGi.

com.ibm.msg.client.osgi.jms< número de versión > .jar
La capa común de código en IBM WebSphere MQ classes for JMS. Para obtener información sobre la arquitectura en capas de las clases WebSphere MQ para JMS, consulte "Una arquitectura en capas" en la página 816.

com.ibm.msg.client.osgi.jms.prereq_< número de versión > .jar
Los archivos JAR (archivo Java) de requisito previo de la capa común.

com.ibm.msg.client.osgi.commonservices.j2se_<version número > .jar
Servicios comunes para aplicaciones Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_< número de versión > .jar
Mensajes para la capa común.

com.ibm.msg.client.osgi.wmq_< número de versión > .jar
El proveedor de mensajería de IBM WebSphere MQ en IBM WebSphere MQ classes for JMS. Para obtener información sobre la arquitectura en capas de IBM WebSphere MQ classes for JMS , consulte "Una arquitectura en capas" en la página 816.

com.ibm.msg.client.osgi.wmq.prereq_< número de versión > .jar
Los archivos JAR de requisito previo para el proveedor de mensajería de IBM WebSphere MQ.

com.ibm.msg.client.osgi.wmq.nls_< número de versión > .jar
Mensajes para el proveedor de mensajería de IBM WebSphere MQ .

com.ibm.mq.osgi.directip_< número de versión > .jar
Los archivos JAR para permitir que el proveedor de mensajería de IBM WebSphere MQ cree una conexión en tiempo real con un intermediario.

donde < número de versión > es el número de versión de WebSphere MQ que se ha instalado.

Los paquetes se instalan en el subdirectorio `java/lib/OSGi` de la instalación de WebSphere MQ o en la carpeta `java\lib\OSGi` en Windows.

El paquete com.ibm.mq.osgi.java < número de versión > .jar, que también se instala en el subdirectorio `java/lib/OSGi` de la instalación de WebSphere MQ , o la carpeta `java\lib\OSGi` en Windows, forma parte de las clases de WebSphere MQ para Java. Este paquete no debe cargarse en un entorno de ejecución OSGi que tenga cargadas las clases WebSphere MQ para JMS.

Los paquetes OSGi para las clases WebSphere MQ para 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 WebSphere 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 tener en cuenta las clases de IBM WebSphere MQ para paquetes JMS, pero los paquetes de IBM WebSphere MQ classes for JMS no tienen en cuenta ningún paquete de usuario. Como resultado, el cargador de clases utilizado en un paquete de IBM WebSphere MQ classes for JMS no puede cargar una clase de salida de canal que esté en un paquete de usuario.

Para obtener más información sobre OSGi, consulte el sitio web de OSGi Alliance.

Resolución de problemas con clases IBM WebSphere MQ para JMS

Puede investigar problemas ejecutando los programas de verificación de la instalación y utilizando los recursos de rastreo y de registro.

Si un programa no se ejecuta correctamente, ejecute uno de los programas de verificación de la instalación, tal como se describe en "La prueba de verificación de instalación punto a punto para las clases WebSphere MQ para JMS" en la página 791 y "La prueba de verificación de la instalación de publicación/suscripción para las clases WebSphere MQ para JMS" en la página 795, y siga los consejos dados en los mensajes de diagnóstico.

Registro yIBM WebSphere MQ classes for JMS

De forma predeterminada, la salida de la función de registro se envía al archivo `mqjms.log`. Puede redirigirla a un archivo o directorio determinado.

Con IBM WebSphere MQ classes for JMS se proporciona la función de registro para notificar errores graves, especialmente problemas que podrían indicar errores de configuración más que errores de programación. De forma predeterminada, la salida de la función de registro se envía al archivo `mqjms.log` en el directorio de trabajo de JVM.

Puede redirigir la salida de la función de registro a otro archivo estableciendo la propiedad com.ibm.msg.client.commonservices.log.outputName. El valor de esta propiedad puede ser:

- Una vía de acceso.
- Una lista de vías de acceso separadas por comas (los datos se registran en todos los archivos).

Cada vía de acceso puede ser:

- Absoluta o relativa.
- `stderr` o `System.err` para representar la salida de errores estándar.
- `sttdout` o `System.out` para representar la salida de estándar.

Si el valor de la propiedad identifica un directorio, la salida de la función de registro se escribe en `mqjms.log` en ese directorio. Si el valor de la propiedad identifica un archivo específico, la salida de la función de registro se escribe en ese archivo.

Puede establecer esta propiedad en el archivo de configuración de IBM WebSphere MQ classes for JMS o como una propiedad del sistema en el mandato **java**. En el ejemplo siguiente, la propiedad se establece como una propiedad del sistema e identifica un archivo específico:

```
java -Djava.library.path=library_path
     -Dcom.ibm.msg.client.commonservices.log.outputName=/mydir/mylog.txt
      MyAppClass
```

En el mandato, *vía_acceso_bibliotecas* es la vía de acceso del directorio donde residen las bibliotecas de IBM WebSphere MQ classes for JMS (consulte “Configuración de las bibliotecas JNI (Java Native Interface)” en la página 739).

Puede inhabilitar la salida de la función de registro estableciendo la propiedad com.ibm.msg.client.commonservices.log.status en OFF. El valor predeterminado de esta propiedad es ON.

Los valores `System.err` y `System.out` se pueden establecer para enviar la salida de registro a las secuencias `System.err` y `System.out` .

Introducción a las clases WebSphere MQ para JMS, para programadores

WebSphere MQ classes for JMS es el proveedor JMS que se proporciona con WebSphere MQ. WebSphere MQ classes for JMS implementa las interfaces definidas en el paquete javax.jms y también proporciona dos conjuntos de extensiones para la API JMS. Las aplicaciones Java Platform, Standard Edition (Java SE) y Java Platform, Enterprise Edition (Java EE) pueden utilizar clases WebSphere MQ para JMS.

La especificación JMS define un conjunto de interfaces que las aplicaciones pueden utilizar para realizar operaciones de mensajería. La versión más reciente de la especificación es la versión 1.1. El paquete javax.jms especifica los detalles de las interfaces JMS y un proveedor JMS implementa estas interfaces para un producto de mensajería específico. WebSphere MQ classes for JMS es un proveedor JMS que implementa las interfaces JMS para WebSphere MQ.

El flujo de lógica dentro de una aplicación JMS empieza con ConnectionFactory y objetos Destination. La aplicación utiliza un objeto ConnectionFactory para crear un objeto Connection, que representa la conexión activa desde la aplicación hasta un servidor de mensajería. La aplicación utiliza el objeto Connection para crear un objeto Session, que es un contexto de hebra única para producir y consumir mensajes. A continuación, la aplicación puede utilizar el objeto Session y un objeto Destination para crear un objeto MessageProducer, que la aplicación utiliza para enviar mensajes al destino especificado. El destino es una cola o un tema del sistema de mensajería y está encapsulado por el objeto Destination. La aplicación también puede utilizar el objeto Session y un objeto Destination para crear un objeto MessageConsumer, que la aplicación utiliza para recibir mensajes que se han enviado al destino especificado.

La especificación JMS espera que los objetos ConnectionFactory y Destination sean objetos administrados. Un administrador crea y mantiene objetos administrados en un repositorio central, y una aplicación JMS recupera estos objetos utilizando JNDI (Java Naming and Directory Interface). El

repositorio de objetos administrados puede ir de un solo archivo a un directorio LDAP (Lightweight Directory Access Protocol).

WebSphere MQ classes for JMS da soporte al uso de objetos administrados. Una aplicación puede utilizar todas las características de WebSphere MQ que se exponen a través de WebSphere MQ classes for JMS sin tener ninguna información específica de WebSphere MQcodificada en la propia aplicación. Esta disposición proporciona a la aplicación un grado de independencia de la configuración subyacente de WebSphere MQ . Para lograr esta independencia, la aplicación puede utilizar JNDI para recuperar fábricas de conexiones y destinos que se almacenan como objetos administrados y utilizar únicamente las interfaces definidas en el paquete javax.jms para realizar operaciones de mensajería. Un administrador puede utilizar la herramienta de administración JMS de WebSphere MQ o IBM WebSphere MQ Explorer para crear y mantener objetos administrados en un repositorio central. Sin embargo, un servidor de aplicaciones normalmente proporciona su propio repositorio para objetos administrados y sus propias herramientas para la creación y mantenimiento de los objetos. Por lo tanto, una aplicación Java EE puede utilizar JNDI para recuperar objetos administrados del repositorio del servidor de aplicaciones o de un repositorio central.

WebSphere MQ classes for JMS también proporciona extensiones a la API JMS. Los releases anteriores de WebSphere MQ classes for JMS contienen extensiones que se implementan en objetos MQConnectionFactory, MQQueue y MQTopic. Estos objetos tienen propiedades y métodos específicos de WebSphere MQ. Los objetos pueden ser objetos administrados, o una aplicación puede crearlos dinámicamente en tiempo de ejecución. Este release de WebSphere MQ classes for JMS mantiene estas extensiones y puede seguir utilizando, sin cambios, cualquier aplicación que utilice estas extensiones. Estas extensiones se conocen como *WebSphere MQ extensiones JMS*. Tenga en cuenta que, en este conjunto de documentación, los objetos que una aplicación crea dinámicamente en tiempo de ejecución *no* se consideran que son objetos administrados.

Además de las extensiones JMS de WebSphere MQ , este release de WebSphere MQ classes for JMS proporciona un conjunto más genérico de extensiones para la API JMS. Estas extensiones se conocen como *extensiones JMS de IBM*y tienen los siguientes objetivos generales:

- Para proporcionar un mayor nivel de coherencia entre los proveedores JMS de IBM
- Para facilitar la escritura de una aplicación puente entre dos sistemas de mensajería IBM
- Para facilitar el puerto de una aplicación desde un proveedor JMS de IBM a otro

El foco principal de estas extensiones afecta a la creación y configuración de fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, pero las extensiones también proporcionan funciones que no están directamente relacionadas con la mensajería, como por ejemplo la función para la determinación de problemas.

Una fábrica de conexiones, una cola o un objeto de tema creado utilizando la interfaz javax.jms o cualquier conjunto de extensiones JMS se puede direccionado utilizando cualquiera de estas API; es decir, se puede convertir a cualquiera de las interfaces. Para mantener la portabilidad de las aplicaciones al máximo nivel, utilice las API más genéricas que se ajusten a sus requisitos.

Las aplicaciones Java SE y Java EE pueden utilizar clases WebSphere MQ para JMS. En la plataforma Java EE , WebSphere MQ classes for JMS da soporte a dos tipos de comunicación entre un componente de una aplicación y un gestor de colas de WebSphere MQ :

Comunicación de salida
Utilizando directamente la API JMS, un componente de aplicación crea una conexión con un gestor de colas y, a continuación, envía y recibe mensajes.

Por ejemplo, el componente de aplicación puede ser un cliente de aplicación, un servlet, una página JavaServer (JSP), un enterprise bean Java (EJB) o un bean controlado por mensajes (MDB). En este tipo de comunicación, el contenedor del servidor de aplicaciones solo proporciona funciones de bajo nivel como soporte de las operaciones de mensajería como, por ejemplo, agrupaciones de conexiones y gestión de hilos.

Comunicación de entrada
Un mensaje que llega a un destino se entrega a un MDB, que a su vez procesa el mensaje.

Las aplicaciones Java EE utilizan MDB para procesar mensajes de forma asíncrona. Un MDB actúa como escucha de mensajes JMS y se implementa mediante un método onMessage(), que define cómo se procesa un mensaje. Un MDB se despliega en el contenedor EJB de un servidor de aplicaciones. La forma precisa en que se configura un MDB depende del servidor de aplicaciones que se utilice, pero la información de configuración debe especificar a qué gestor de colas debe conectarse, cómo conectarse al mismo, qué destino debe supervisar mensajes y el comportamiento transaccional del MDB. El contenedor EJB utilizará esta información. Cuando un mensaje que satisface los criterios de selección del MDB llega al destino especificado, el contenedor EJB utiliza WebSphere MQ classes for JMS para recuperar el mensaje del gestor de colas y, a continuación, entrega el mensaje al MDB llamando a su método onMessage().

Clases IBM WebSphere MQ para la arquitectura JMS

Las clases IBM WebSphere MQ para JMS, tal como se proporcionan en IBM WebSphere MQ Versión 7.0y releases posteriores, contienen una serie de mejoras en comparación con los releases anteriores. Algunas de estas mejoras son el resultado de cambios en la implementación de las clases IBM WebSphere MQ para JMS, y algunas son el resultado de que las clases IBM WebSphere MQ para JMS explotan los cambios en la función IBM WebSphere MQ subyacente.

En las secciones siguientes se resumen las mejoras clave.

Una arquitectura en capas

En releases anteriores de WebSphere MQ, la implementación de WebSphere MQ classes for JMS ha sido totalmente específica de WebSphere MQ. Otros productos IBM que proporcionan sistemas de mensajería también han incluido proveedores JMS, pero estos proveedores JMS tienen muy poco o nada en común con la implementación de clases WebSphere MQ para JMS.

A partir de WebSphere MQ V7.0, WebSphere MQ classes for JMS tiene una arquitectura en capas. La capa superior de código es una capa común que puede utilizar cualquier proveedor JMS de IBM . Cuando una aplicación llama a un método JMS, cualquier proceso de la llamada que no sea específico de un sistema de mensajería lo realiza la capa común, que también proporciona una respuesta coherente a la llamada. Todo procesamiento de la llamada que sea específico de un sistema de mensajería se delega en una capa inferior. La Figura 125 en la página 816 muestra la arquitectura en capas.

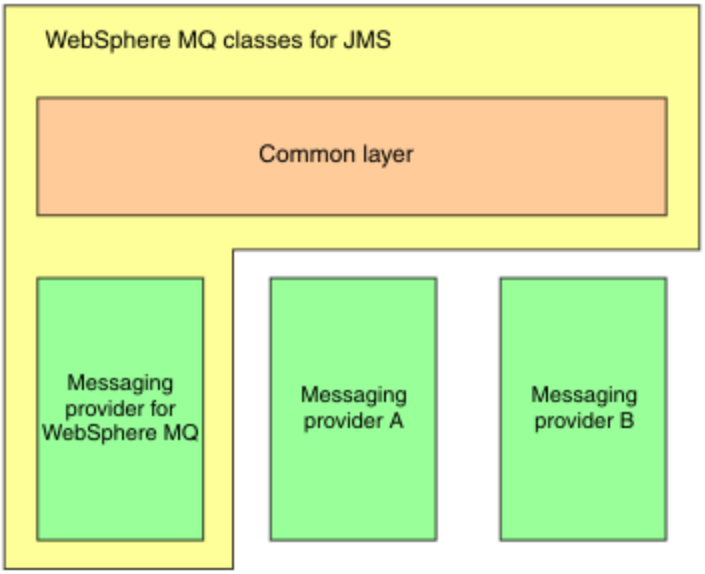


Figura 125. La arquitectura en capas para proveedores JMS de IBM

El paso a una arquitectura en capas tiene los siguientes objetivos:

- Para mejorar la coherencia del comportamiento de los diversos proveedores JMS de IBM

- Para facilitar la escritura de una aplicación puente entre dos sistemas de mensajería IBM
- Para facilitar el puerto de una aplicación desde un proveedor JMS de IBM a otro

Esta implementación de WebSphere MQ classes for JMS también presenta un nuevo conjunto de extensiones para la API JMS. Estas extensiones se conocen como *extensiones JMS de IBM*. El foco principal de estas extensiones se refiere a la creación y configuración dinámica de fábricas de conexiones y destinos en tiempo de ejecución.

Una aplicación que utiliza las extensiones JMS de IBM se inicia creando un objeto de fábrica JmsFactory, especificando como parámetro una constante que identifica el sistema de mensajería elegido. La aplicación utiliza el objeto JmsFactoryFactory para crear fábricas de conexiones y destinos que tengan las clases especializadas correctas para el sistema de mensajería elegido.

A continuación, la aplicación puede configurar las fábricas de conexiones y destinos definiendo sus propiedades. Las extensiones JMS de IBM proporcionan un conjunto de métodos para establecer propiedades. Estos métodos son independientes de cualquier sistema de mensajería. Cada tipo de datos tiene su propio método set y cada propiedad se identifica con un nombre, que se define como un miembro final estático de la clase WMQConstants. Cuando una aplicación invoca uno de estos métodos, uno de los parámetros de la llamada es el nombre de la propiedad y el otro parámetro es el valor de la propiedad.

Por ejemplo, si WebSphere MQ es el sistema de mensajería, una de las propiedades de una fábrica de conexiones es el nombre del gestor de colas al que conectarse. Utilizando las extensiones JMS de IBM , una aplicación establece el nombre del gestor de colas en JUPITER llamando al método siguiente:

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

Por el contrario, una aplicación puede realizar la misma función llamando al método siguiente:

```
MQConnectionFactory myCF;
...
myCF.setQueueManager("JUPITER");
```

Este método es una extensión JMS de WebSphere MQ y es específico de WebSphere MQ como sistema de mensajería. Por lo tanto, el uso de este método hace que la aplicación sea potencialmente menos fácil de portar a otro proveedor JMS de IBM .

La relación entre las clases WebSphere MQ para JMS y las clases WebSphere MQ para Java

En los releases de WebSphere MQ, anteriores a la versión 7.0, las clases WebSphere MQ para JMS se implementaban casi por completo como una capa de código sobre las clases WebSphere MQ para Java. Esta disposición ha causado cierta confusión entre los desarrolladores de aplicaciones porque establecer campos o llamar a métodos en la clase MQEnvironment puede provocar efectos no deseados e inesperados en el comportamiento en tiempo de ejecución del código que se escribe utilizando WebSphere MQ classes for JMS. Además, la implementación de las clases WebSphere MQ para JMS tenía algunas restricciones en áreas en las que la API JMS no es un ajuste natural sobre las clases WebSphere MQ para Java, y estas restricciones han dado lugar a algunos problemas relacionados con el rendimiento del tiempo de ejecución.

A partir de WebSphere MQ V7.0, la implementación de las clases WebSphere MQ para JMS ya no depende de las clases WebSphere MQ para Java. WebSphere MQ classes for Java y WebSphere MQ classes for JMS ahora son iguales que utilizan una interfaz Java común a la MQI. Esta disposición permite más ámbito para optimizar el rendimiento, y significa que el establecimiento de campos o métodos de llamada en la clase MQEnvironment no tiene ningún efecto en el comportamiento en tiempo de ejecución del código que se escribe utilizando WebSphere MQ classes for JMS. Figura 126 en la página 818 muestra la relación entre las clases WebSphere MQ para JMS y las clases WebSphere MQ para Java en releases anteriores de WebSphere MQ y en WebSphere MQ V7.0 y releases posteriores.

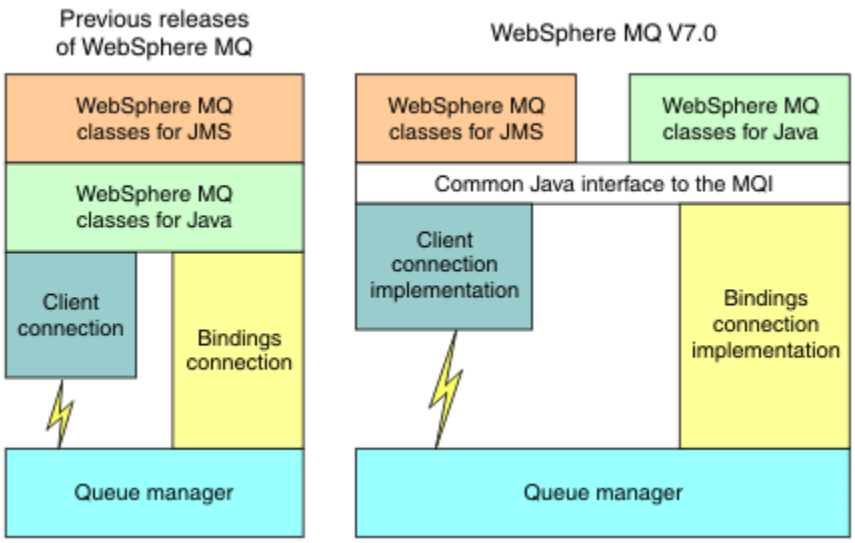


Figura 126. La relación entre las clases WebSphere MQ para JMS y las clases WebSphere MQ para Java

Para mantener la compatibilidad con releases anteriores, las clases de salida de canal escritas en Java pueden seguir utilizando las clases WebSphere MQ para interfaces Java, incluso si se llama a las clases de salida de canal desde WebSphere MQ para JMS. Sin embargo, el uso de las clases WebSphere MQ para interfaces Java significa que las aplicaciones siguen dependiendo de las clases WebSphere MQ para el archivo JAR Java, com.ibm.mq.jar. Si no desea com.ibm.mq.jar en la vía de acceso de clases, puede utilizar el nuevo conjunto de interfaces en el paquete com.ibm.mq.exits en su lugar.

Ahora puede crear y configurar objetos administrados JMS con WebSphere MQ Explorer.

Mensajería de publicación/suscripción

WebSphere MQ V7.0, y los releases posteriores, contienen la función de publicación/suscripción incorporada. Esta función sustituye a WebSphere MQ Publish/Subscribe, que se suministró con WebSphere MQ V6.0.

Las clases WebSphere MQ para aplicaciones JMS pueden utilizar la función de publicación/suscripción incorporada, y pueden utilizarla en lugar de utilizar WebSphere Event Broker o WebSphere Message Broker para mensajería de publicación/suscripción con WebSphere MQ como transporte. Configurar WebSphere MQ classes for JMS para que utilice la nueva función es más sencillo que configurar WebSphere MQ classes for JMS para que utilice WebSphere MQ Publish/Subscribe, WebSphere Event Broker o WebSphere Message Broker. Los administradores y desarrolladores de aplicaciones ya no necesitan gestionar colas de publicación, colas de suscriptores, almacenes de suscripción ni limpieza de suscriptores. Además, los objetos ConnectionFactory y Topic tienen menos propiedades.

La función de publicación/suscripción incorporada también proporciona algunas características adicionales, como por ejemplo, las publicaciones retenidas y dos esquemas de comodines para especificar un rango de temas a los que desea suscribirse una aplicación.

Una aplicación puede seguir utilizando una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker para la mensajería de publicación/suscripción. Este soporte no ha cambiado.

Las aplicaciones que utilizan WebSphere MQ Publicación/Suscripción pueden utilizar la función de publicación/suscripción incorporada sin cambios cuando se actualiza el gestor de colas al que están conectadas. Se hace caso omiso de las propiedades establecidas por una aplicación, pero que no son necesarias para la función de publicación/suscripción incorporada.

Proveedor de mensajería de WebSphere MQ

El proveedor de mensajería de WebSphere MQ tiene dos modalidades de operación:

- *WebSphere MQ modalidad normal de proveedor de mensajería*
- *WebSphere MQ modalidad de migración de proveedor de mensajería*

La modalidad normal del proveedor de mensajería de WebSphere MQ utiliza todas las características de WebSphere MQ Versión 7.0y posteriores gestores de colas de release para implementar JMS. Esta modalidad sólo se utiliza para conectarse a un gestor de colas de WebSphere MQ y puede conectarse a WebSphere MQ Versión 7.0y a los gestores de colas de release posteriores en modalidad de cliente o de enlaces. Esta modalidad se optimiza para utilizar la nueva WebSphere MQ Versión 7.0 y la función de release posterior.

La modalidad de migración del proveedor de mensajería de WebSphere MQ se basa en la función WebSphere MQ Versión 6.0 y sólo utiliza las características que estaban disponibles en el gestor de colas de WebSphere MQ Versión 6.0 para implementar JMS. Puede conectarse a un WebSphere MQ Versión 7.0 y posteriores gestores de colas de release utilizando la modalidad de migración de proveedor de mensajería de WebSphere MQ , pero no puede utilizar ninguna de las optimizaciones de la versión 7.0 . Esta modalidad permite realizar conexiones a cualquiera de las siguientes versiones de gestores de colas:

1. WebSphere MQ Versión 7.0y posterior, gestor de colas en modalidad de enlaces o cliente, pero esta modalidad sólo utiliza las características que estaban disponibles para un gestor de colas WebSphere MQ Versión 6.0
2. WebSphere MQ Versión 6.0 o un gestor de colas anterior en modalidad de cliente

Si desea conectarse a WebSphere Event Broker o WebSphere Message Broker utilizando WebSphere MQ Enterprise Transport, utilice la modalidad de migración de proveedor de mensajería de WebSphere MQ . Si utiliza WebSphere MQ Real-Time Transport, la modalidad de migración de proveedor de mensajería de WebSphere MQ se selecciona automáticamente, porque ha seleccionado explícitamente propiedades en el objeto de fábrica de conexiones. La conexión con WebSphere Event Broker o WebSphere Message Broker utilizando WebSphere MQ Enterprise Transport sigue las reglas generales para la selección de modalidad descritas en Reglas para seleccionar la modalidad de proveedor de mensajería de WebSphere MQ .

Consumo de mensajes asíncronos

WebSphere MQ V7.0 y cualquier release posterior da soporte al consumo de mensajes asíncronos. Una aplicación puede registrar una función de devolución de llamada para un destino. Cuando se envía un mensaje adecuado al destino, WebSphere MQ llama a la función y pasa el mensaje como un parámetro. A continuación, la función procesa el mensaje de forma asíncrona. En releases anteriores de WebSphere MQ, esta característica solo estaba disponible cuando se utilizaban clases WebSphere MQ para JMS.

WebSphere MQ classes for JMS se ha modificado para aprovechar esta nueva característica en WebSphere MQ V7.0 y cualquier release posterior. La implementación de escuchas de mensajes JMS es ahora un ajuste más natural con WebSphere MQ, y WebSphere MQ classes for JMS ya no tiene que sondear un destino para comprobar si se ha enviado un mensaje adecuado al destino. El rendimiento de los escuchas de mensajes JMS se mejora como resultado, especialmente cuando una aplicación utiliza varios escuchas de mensajes en una sesión para supervisar varios destinos. El rendimiento de mensajes aumenta y el tiempo que se tarda en entregar un mensaje a un escucha de mensajes después de que haya llegado a un destino se reduce.

Los beans controlados por mensajes (MDB) tienen unas mejoras de rendimiento parecidas. Además, debido a otra mejora de la función WebSphere MQ , varios MDB que están consumiendo mensajes del mismo destino ahora experimentan una contención reducida en los mensajes.

Selección de mensajes

Con la excepción de seleccionar mensajes por identificador de mensaje o identificador de correlación, todas las selecciones de mensajes en los releases de WebSphere MQ anteriores a la versión 7.0 las realizaban las clases WebSphere MQ para JMS. En WebSphere MQ V7.0, y cualquier release posterior, el gestor de colas realiza toda la selección de mensajes.

Como resultado, aumenta la productividad de los mensajes para las aplicaciones que consumen mensajes utilizando la selección de mensajes. La mejora de rendimiento es mayor para una aplicación que se conecta en modalidad de cliente porque sólo los mensajes que cumplen los criterios de selección se transportan a través de la red, y WebSphere MQ classes for JMS sólo maneja los mensajes que entrega a la aplicación.

Compartición de una conexión de comunicaciones

En releases anteriores de WebSphere MQ, si una aplicación cliente WebSphere MQ se conectaba a un gestor de colas más de una vez utilizando el mismo canal MQI, cada instancia del canal MQI necesitaba una conexión TCP independiente. En WebSphere MQ V7.0 y cualquier release posterior, cada conexión con el gestor de colas que utilice el mismo canal MQI puede compartir una única conexión TCP. Esta disposición significa que se necesitan menos recursos de red y el tiempo total que se tarda en crear varias conexiones con el gestor de colas se reduce, especialmente cuando se utiliza SSL porque el reconocimiento SSL sólo tiene lugar una vez al inicio de la conexión TCP.

WebSphere MQ classes for JMS aprovecha esta mejora. Para una aplicación que se conecta a un gestor de colas en modalidad de cliente, las clases WebSphere MQ para JMS pueden crear más de una conexión con un gestor de colas utilizando el canal MQI con el nombre especificado como propiedad del objeto ConnectionFactory . Estas conexiones al gestor de colas ahora pueden compartir una sola conexión TCP.

Lectura anticipada en conexiones de cliente

Si una aplicación utiliza una conexión de cliente para consumir mensajes no persistentes de un destino, el destino se puede configurar de modo que WebSphere MQ classes for JMS utilice un almacenamiento intermedio para almacenar los mensajes de interés antes de entregarlos a la aplicación. Esta optimización se llama *de lectura anticipada* y puede ser utilizada por aplicaciones que consumen mensajes de forma asíncrona invocando el método receive(), y mediante escuchas de mensajes y MDBs, que consumen mensajes de forma asíncrona. La lectura anticipada es especialmente efectiva en destinos con un gran número de mensajes que hay que consumir rápidamente.

La lectura anticipada no se aplica a los mensajes permanentes porque si los mensajes permanentes se leen y almacenan en un almacenamiento intermedio, el gestor de colas ya no podrá recuperar dichos mensajes tras un error. No obstante, una aplicación que consuma mensajes de un destino con una combinación de mensajes permanentes y no permanentes puede utilizar la lectura anticipada. Se mantiene el orden de los mensajes, pero las ventajas de la lectura anticipada de la ejecución solo se aplican a los mensajes no permanentes.

Al decidir si se debe utilizar la lectura anticipada, tenga en cuenta los puntos siguientes:

- Si una aplicación consume mensajes de un destino configurado para la lectura anticipada y, por algún motivo, la aplicación termina, se descartarán los mensajes no permanentes almacenados en ese momento en el almacenamiento intermedio.
- Si se cumplen todas las condiciones siguientes, podría ocurrir que los mensajes enviados a una cola en una sesión no se recibieran en el orden en que se han enviado:
  - La aplicación utiliza dos consumidores de mensajes en la misma sesión para consumir los mensajes de la cola.
  - Cada consumidor de mensajes utiliza un objeto de destino distinto para la cola.
  - Uno de los objetos de destino, o ambos, están configurados para la lectura anticipada.

Envío de mensajes

Cuando una aplicación envía mensajes a un destino, el destino se puede configurar de modo que, cuando la aplicación llama a send (), WebSphere MQ classes for JMS reenvía el mensaje al gestor de colas y devuelve el control a la aplicación sin determinar si el gestor de colas ha recibido el mensaje de forma segura. Las clases WebSphere MQ para JMS solo pueden funcionar de esta forma para los mensajes no persistentes y para los mensajes persistentes enviados en una sesión con transacción.

Para los mensajes persistentes enviados en una sesión de transacción, la aplicación finalmente determina si el gestor de colas ha recibido los mensajes de forma segura cuando llama a commit (). Para los mensajes enviados en una sesión que no es transaccional, la propiedad SENDCHECKCOUNT del objeto ConnectionFactory especifica cuántos mensajes deben enviarse antes de que WebSphere MQ classes for JMS compruebe que el gestor de colas ha recibido los mensajes de forma segura.

Esta optimización es sumamente beneficiosa para las aplicaciones que se conectan a un gestor de colas en modalidad cliente y necesitan enviar una secuencia de mensajes en sucesión rápida, pero no requieren una reacción inmediata del gestor de colas por cada mensaje enviado.

Salidas de canal

Cuando se llama desde WebSphere MQ classes for JMS, los programas de salida de canal escritos en C o C++ se comportan ahora de la misma forma que cuando se llaman desde un cliente MQI de Websphere MQ . Se ha mejorado el rendimiento de las clases de salida de canal escritas en Java y ahora puede escribir clases de salida de canal utilizando un nuevo conjunto de interfaces en el paquete com.ibm.mq.exits en lugar de utilizar las interfaces en las clases WebSphere MQ para Java.

Propiedades del mensaje

Un mensaje JMS consta de un conjunto de campos de cabecera, un conjunto de propiedades y un cuerpo que contiene los datos de aplicación. Como mínimo, un mensaje de WebSphere MQ consta de un descriptor de mensaje y los datos de la aplicación.

Cuando una aplicación WebSphere MQ classes for JMS envía un mensaje JMS, WebSphere MQ classes for JMS correlaciona el mensaje JMS en un mensaje WebSphere MQ . Algunos de los campos y propiedades de cabecera JMS se correlacionan en campos en el descriptor de mensaje, y algunos se correlacionan en campos en una cabecera WebSphere MQ adicional denominada una cabecera MQRFH2 . Cuando una aplicación WebSphere MQ classes for JMS recibe un mensaje JMS, WebSphere MQ classes for JMS realiza la correlación inversa.

Por lo tanto, una aplicación que utiliza MQI para recibir mensajes de una aplicación WebSphere MQ para JMS debe poder manejar una cabecera MQRFH2 . Si la aplicación no puede manejar una cabecera MQRFH2 , la propiedad TARGCLIENT del objeto Destination se puede establecer para indicar a WebSphere MQ classes for JMS que no incluya una cabecera MQRFH2 en los mensajes WebSphere MQ . Sin embargo, al excluir la cabecera MQRFH2 , se pierde la información contenida en algunos de los campos y propiedades de cabecera JMS.

De forma similar, una aplicación que utiliza MQI para enviar mensajes a una aplicación WebSphere MQ classes for JMS debe incluir una cabecera MQRFH2 en cada mensaje. Si no se incluye una cabecera MQRFH2 , WebSphere MQ classes for JMS sólo puede establecer los campos de cabecera JMS y las propiedades que se pueden derivar de los campos de un descriptor de mensaje.

WebSphere MQ V7.0 proporciona algún soporte adicional para las aplicaciones que utilizan MQI para recibir mensajes de, y enviar mensajes a, WebSphere MQ clases para aplicaciones JMS.

Cuando una aplicación llama a MQGET para recibir un mensaje de una aplicación WebSphere MQ para JMS, la aplicación puede optar por recibir el mensaje de una de las maneras siguientes:

1. El mensaje se entrega con un descriptor de mensaje, una cabecera MQRFH2 que contiene datos derivados de campos y propiedades de cabecera JMS y los datos de aplicación.
2. El mensaje se entrega con un descriptor de mensaje, los datos de aplicación y un conjunto de propiedades de mensajes.

En la opción 2, cada propiedad de mensaje representa un campo o propiedad de cabecera JMS correlacionado originalmente por WebSphere MQ classes for JMS en un campo de una cabecera MQRFH2 . Después de la llamada MQGET, la aplicación puede utilizar la llamada a MQINQMP para obtener los valores de las propiedades de mensajes. Si se utiliza la opción 2 en lugar de la opción 1 para recibir un mensaje, se simplificará la lógica de la aplicación de la siguiente manera:

- La aplicación no tiene que analizar la parte variable de la cabecera MQRFH2 , que contiene el campo de cabecera JMS y los datos de propiedad codificados en un formato de tipo XML.

- La aplicación no tiene que convertir los datos de caracteres en la parte variable de la cabecera MQRFH2.

En consecuencia, antes de que una aplicación llame a MQPUT para enviar un mensaje a una aplicación WebSphere MQ para JMS, la aplicación puede utilizar la llamada MQSETMP para establecer los valores de las propiedades del mensaje en lugar de construir una cabecera MQRFH2 .

Capacidad de servicio

WebSphere MQ classes for JMS contiene una serie de mejoras relacionadas con la capacidad de servicio:

- Rastreo.

  WebSphere MQ classes for JMS contiene una clase que una aplicación puede utilizar para controlar el rastreo. Una aplicación puede iniciar y detener el rastreo, especificar el nivel de detalle necesario en un rastreo y personalizar la salida de rastreo de varias formas.
- Registro.

  WebSphere MQ classes for JMS mantiene un archivo de registro, que contiene mensajes sobre los errores que debe corregir. Los mensajes están escritos en texto plano. WebSphere MQ classes for JMS contiene una clase que una aplicación puede utilizar para especificar la ubicación del archivo de registro y su tamaño máximo.
- Tecnología de soporte de primera anomalía (FFST).

  Si se produce una anomalía grave, WebSphere MQ classes for JMS genera un informe FFST en un archivo FDC. El informe FFST contiene información que el servicio de IBM puede utilizar para diagnosticar el problema más rápidamente.
- Información de versión.

  WebSphere MQ classes for JMS contiene una clase que una aplicación puede utilizar para consultar la versión de WebSphere MQ classes for JMS.
- Mensajes de excepción.

  Los mensajes de excepción se han mejorado para proporcionar más información sobre las causas de los errores y las acciones necesarias para corregir los errores.
- Servidores de aplicaciones.

  Se ha mejorado la integración de las características de capacidad de servicio de las clases WebSphere MQ para JMS con las de WebSphere Application Server.

MQC se sustituye por MQConstants

Se proporciona un nuevo paquete, com.ibm.mq.constants, con IBM WebSphere MQ Versión 7.0. Este paquete contiene la clase MQConstants, que implementa un número de interfaces. MQConstants contiene definiciones de todas las constantes que estaban en la interfaz MQC y un número de constantes nuevas. Las interfaces de este paquete siguen de cerca los nombres de los archivos de cabecera de constantes utilizados en IBM WebSphere MQ.

Por ejemplo, la interfaz CMQC contiene una constante MQOO_INPUT_SHARED; estas interfaz y esta constante se corresponden con el archivo de cabecera `cmqc.h` y la constante MQOO_INPUT_SHARED.

com.ibm.mq.constants se puede utilizar con las clases IBM WebSphere MQ para Java y IBM WebSphere MQ para JMS.

MQC sigue estando presente y tiene las constantes que tenía antes. No obstante, deberá utilizarse el nuevo paquete com.ibm.mq.constants en las aplicaciones nuevas.

Escritura de clases de WebSphere MQ para aplicaciones JMS

Después de una breve introducción al modelo JMS, este tema proporciona una guía detallada sobre cómo escribir clases de WebSphere MQ para aplicaciones JMS.

El modelo JMS

El modelo JMS define un conjunto de interfaces que las aplicaciones Java pueden utilizar para realizar operaciones de mensajería. WebSphere MQ classes for JMS, como proveedor de JMS, define cómo están relacionados los objetos JMS con los conceptos de WebSphere MQ . La especificación JMS espera que determinados objetos JMS sean objetos administrados.

La especificación JMS y el paquete javax.jms definen un conjunto de interfaces que las aplicaciones Java pueden utilizar para realizar operaciones de mensajería. La lista siguiente resume las interfaces JMS principales:

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.

Sesión (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 127 en la página 823 muestras estos objetos y sus relaciones.

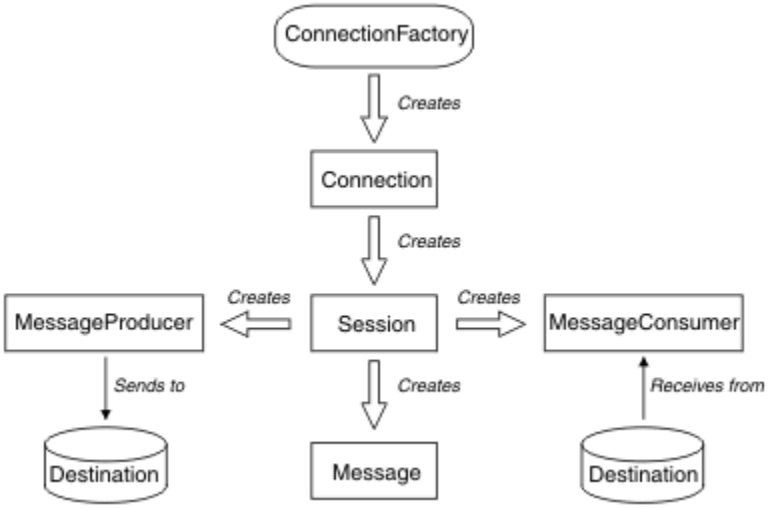


Figura 127. Objetos JMS y sus relaciones

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.

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.

Con las versiones de JMS anteriores a JMS 1.1, la programación para el dominio punto a punto utiliza un conjunto de interfaces y métodos, y la programación para el dominio de publicación/suscripción utiliza otro conjunto. Los dos conjuntos son similares, pero independientes. Con 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 103 en la página 824 lista las interfaces independientes de dominio JMS y sus interfaces específicas de dominio correspondientes.

Tabla 103. Interfaces independientes del dominio JMS y específicas del dominio

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
Sesión (Session).	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 1.1 retiene todas las interfaces específicas del dominio y, por lo tanto, las aplicaciones existentes pueden seguir utilizando estas interfaces. Sin embargo, para las aplicaciones nuevas, considere la posibilidad de utilizar las interfaces independientes del dominio.

En las clases de WebSphere MQ para JMS, los objetos JMS están relacionados con los conceptos de WebSphere 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 manejador de conexiones WebSphere MQ , que por lo tanto define el ámbito transaccional de la sesión.
- Un objeto MessageProducer y un objeto MessageConsumer encapsulan cada uno un descriptor de contexto de objeto WebSphere MQ .

Cuando se utilizan las clases WebSphere MQ para JMS, se aplican todas las reglas normales de WebSphere 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 objetos administrados en un repositorio central, y una aplicación JMS recupera estos objetos utilizando JNDI (Java Naming and Directory Interface).

En WebSphere MQ classes for JMS, 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 las clases WebSphere MQ para JMS, los objetos de los tipos siguientes pueden ser objetos administrados:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Cola
- Tema
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Mensajes JMS

Los mensajes JMS se componen de una cabecera, propiedades y un cuerpo. JMS define cinco tipos de cuerpo de mensaje.

Los mensajes JMS se componen 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.

Body (Cuerpo)
JMS define varios tipos de cuerpo de mensaje que cubren la mayoría de estilos de mensajería actualmente en uso.

JMS define cinco tipos de cuerpo de mensaje:

Corriente de datos (Stream)
Una corriente de valores primitivos 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 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 Java serializable

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 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 a una aplicación, utilizando selectores de mensajes, que un proveedor JMS seleccione o filtre mensajes en su nombre, 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. Sin embargo, los desarrolladores de aplicaciones deben tener en cuenta que los proveedores JMS probablemente manejan los datos en un cuerpo de mensaje de forma más eficiente que los datos en 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 JMS permite a un cliente especificar los mensajes en los que está interesado utilizando la cabecera de 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.

Un selector 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'. Al igual que los literales de serie Java, estos utilizan la codificación de caracteres Unicode.
  - Un literal numérico exacto es un valor numérico sin coma decimal, tal como 57, -957 y +62. Los números en el rango de Java long están soportados.
  - 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. Los números en el rango de Java doble están soportados.
  - 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 es el mismo que se ha definido para Java: espacio, tabulador horizontal, salto de página 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 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.
  - La edad NO ENTRE 15 y 19 es equivalente a la 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.

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"
```

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.

Aunque SQL da soporte a la comparación decimal fija y a la aritmética, los selectores de mensajes JMS 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.

Correlación de mensajes JMS con mensajes WebSphere MQ

Los mensajes de WebSphere MQ se componen de un descriptor de mensaje, una cabecera MQRFH2 opcional y un cuerpo. El contenido de un mensaje JMS se correlaciona parcialmente y se copia parcialmente en un mensaje de WebSphere MQ .

En este tema se describe cómo la estructura de mensajes JMS que se describe en la primera parte de esta sección se correlaciona con un mensaje WebSphere MQ . Es de interés para los programadores que desean transmitir mensajes entre JMS y aplicaciones WebSphere MQ tradicionales. También es de interés para las personas que desean manipular mensajes transmitidos entre dos aplicaciones JMS, por ejemplo, en una implementación de WebSphere Message Broker.

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, todas las comunicaciones se realizan directamente a través de TCP/IP; no hay implicados mensajes ni colas de WebSphere MQ .

Los mensajes de WebSphere MQ constan de tres componentes:

- WebSphere MQ Message Descriptor (MQMD)
- Una cabecera WebSphere MQ MQRFH2
- El cuerpo del mensaje.

MQRFH2 es opcional y su inclusión en un mensaje de salida se controla mediante un distintivo en la clase de destino JMS. Puede establecer este distintivo utilizando la herramienta de administración JMS de WebSphere MQ . Puesto que MQRFH2 contiene información específica de JMS, inclúyalo siempre en el mensaje cuando el remitente sepa que el destino de recepción es una aplicación JMS. Normalmente, omita MQRFH2 al enviar un mensaje directamente a una aplicación no JMS. Esto se debe a que una aplicación de este tipo no espera un mensaje MQRFH2 en su mensaje WebSphere 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 128 en la página 829 muestra cómo se transforma la estructura de un mensaje JMS en un mensaje de WebSphere MQ y de nuevo:

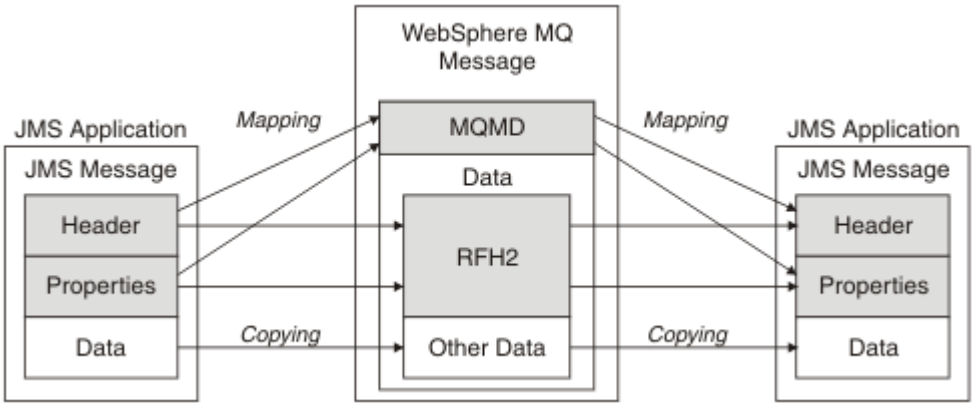


Figura 128. Cómo se transforman los mensajes entre JMS y WebSphere MQ utilizando la cabecera MQRFH2

Las estructuras se transforman de dos modos:

Correlación
Cuando el MQMD incluye un campo que es equivalente al campo JMS, el campo JMS se correlaciona con el campo MQMD. Los campos MQMD adicionales se exponen como propiedades JMS, porque es posible que una aplicación JMS tenga que obtener o establecer estos campos al comunicarse con una aplicación no JMS.

Copia
Cuando no hay ningún MQMD equivalente, se pasa un campo o propiedad de cabecera JMS, posiblemente transformado, 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 versión 2 de MQRFH2 es una cabecera extensible y también puede transportar información adicional que no está directamente asociada con JMS. Sin embargo, esta sección sólo cubre su uso 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 modela en el patrón de cabecera *estándar* WebSphere 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 (valor: "R","F","H"," ") también está definido.

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 104. Valores posibles para el campo NameValueCCSID

Valor	Significado
1200	UCS2 abierto
1208	UTF8
13488	Subconjunto UCS2 2.0
17584	Subconjunto UCS2 2.1 (incluye el símbolo del Euro)

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:

La 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 WebSphere Message Broker. Describe el dominio, formato, tipo y conjunto de mensajes de un mensaje.

Tabla 105. 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`.

La 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 `MQRFH2` de JMS.

La 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.

La carpeta < mqext>

`mqext` contiene propiedades que sólo utiliza WebSphere Application Server. La carpeta sólo está presente si la aplicación ha establecido al menos una de las propiedades definidas de IBM .

Tabla 106. 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>

No añada sus propias propiedades en la carpeta mqext.

La carpeta < mqps>

mqps contiene propiedades que solo son utilizadas por la publicación/suscripción de IBM WebSphere MQ. La carpeta sólo está presente si la aplicación ha establecido al menos una de las propiedades de publicación/suscripción integradas.

Tabla 107. 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>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

No añada sus propias propiedades en la carpeta mqps.

En la Tabla 108 en la página 832 se muestra una lista completa de los nombres de las propiedades.

Tabla 108. Carpetas y propiedades de MQRFH2 utilizadas por JMS				
Nombre de campo JMS	**Tipo Java**	**Nombre de carpeta de MQRFH2**	**Nombre de propiedad**	**Tipo/valores**
JMSDestination	Destino	jms	Dst	serie
JMSExpiration	largo	jms	Exp	i8
JMSPriority	int	jms	Pri	i4

Tabla 108. Carpetas y propiedades de MQRFH2 utilizadas por JMS (continuación)				
Nombre de campo JMS	**Tipo Java**	**Nombre de carpeta de MQRFH2**	**Nombre de propiedad**	**Tipo/valores**
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Serie	jms	Cid	serie
JMSReplyTo	Destino	jms	Rto	serie
JMSTimestamp	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 108 en la página 832.

`datatype`
debe coincidir, después de agrupar los datos, con uno de los tipos de datos listados en la Tabla 109 en la página 834.

`value`
es una representación de tipo serie del valor que se debe transmitir, utilizando las definiciones de la Tabla 109 en la página 834.

Un valor nulo se codifica utilizando la sintaxis siguiente:

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

No utilice `xsi:nil='false'`.

Tabla 109. 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 >
- `&apos;` para el carácter '
- `&quot;` para el carácter "

Campos JMS y propiedades con los campos MQMD correspondientes
Estas tablas muestran los campos MQMD equivalentes a los campos de cabecera JMS, las propiedades JMS y las propiedades específicas del proveedor JMS.

Tabla 110 en la página 835 lista los campos de cabecera JMS y Tabla 111 en la página 835 lista las propiedades JMS que se correlacionan directamente con los campos MQMD. Tabla 112 en la página 835 lista las propiedades específicas del proveedor y los campos MQMD con los que están correlacionados.

Tabla 110. Correlación de campos de cabecera JMS con campos MQMD			
Campo de cabecera JMS	**Tipo Java**	**Campo de MQMD**	**Tipo C**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	largo	Caducidad	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	Cadena	MsgID	MQBYTE24
JMSTimestamp	largo	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Cadena	CorrelId	MQBYTE24

Tabla 111. Correlación de propiedades JMS con campos MQMD			
Propiedad JMS	**Tipo Java**	**Campo de MQMD**	**Tipo C**
JMSXUserID	Cadena	UserIdentifier	MQCHAR12
JMSXAppID	Cadena	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Cadena	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabla 112. Correlación de propiedades específicas del proveedor JMS con campos MQMD			
Propiedad específica del proveedor JMS	**Tipo 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
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	Cadena	Format “1” en la página 836	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codificación	MQLONG

Tabla 112. Correlación de propiedades específicas del proveedor JMS con campos MQMD (continuación)

Propiedad específica del proveedor JMS	Tipo Java	Campo de MQMD	Tipo C
JMS_IBM_Character_Set	Cadena	CodedCharacterSetId"2" en la página 836	MQLONG
JMS_IBM_PutDate	Cadena	PutDate	MQCHAR8
JMS_IBM_PutTime	Cadena	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 definir mediante la aplicación que establece la propiedad JMS_IBM_Format del mensaje (tenga en cuenta que hay un límite de 8 caracteres), o puede tomar de forma predeterminada el formato WebSphere MQ del cuerpo del mensaje adecuado al tipo de mensaje 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 serie que contiene el equivalente de juego de caracteres Java para el valor numérico CodedCharacterSetId . El campo MQMD CodedCharacterSetId es un valor numérico que contiene el equivalente de la serie de juego de caracteres Java especificada por la propiedad JMS_IBM_Character_Set.

Correlación de campos JMS en campos WebSphere MQ (mensajes salientes)
Estas tablas muestran cómo se correlacionan los campos de cabecera y propiedad JMS en los campos MQMD y MQRFH2 en el momento de enviar () o publicar ().

La Tabla 113 en la página 836 muestra cómo se correlacionan los campos de cabecera JMS en campos MQMD/RFH2 en el momento de enviar () o publicar (). Tabla 114 en la página 837 muestra cómo se correlacionan las propiedades JMS en los campos MQMD/RFH2 en el momento de enviar () o publicar (). Tabla 115 en la página 837 muestra cómo se correlacionan las propiedades específicas del proveedor JMS con los campos MQMD en el momento de enviar () o publicar (),

Para los campos marcados como Objeto de mensaje, el valor transmitido es el valor contenido en el mensaje JMS inmediatamente antes de la operación send () o publish (). La operación no modifica el valor del mensaje JMS.

Para los campos marcados como establecidos por el método de envío, se asigna un valor cuando se realiza send () o publish () (se ignora cualquier valor contenido en el mensaje JMS). El valor del mensaje 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 113. Correlación de los campos de mensajes salientes

Nombre de campo de cabecera 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	Priority	MQRFH2	Método Send
JMSMessageID	MsgID		Método Send

Tabla 113. Correlación de los campos de mensajes salientes (continuación)

Nombre de campo de cabecera JMS	Campo de MQMD utilizado para la transmisión	Cabecera	Establecido por
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 MQMD CodedCharacterSetId es un valor numérico que contiene el equivalente de la serie de juego de caracteres Java especificada por la propiedad JMS_IBM_Character_Set.

Tabla 114. Correlación de propiedades JMS de mensaje de salida

Nombre de la propiedad 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

Tabla 115. Correlación de propiedades específicas del proveedor JMS de salida

Nombre de propiedad específico del proveedor 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

Tabla 115. Correlación de propiedades específicas del proveedor JMS de salida (continuación)			
Nombre de propiedad específico del proveedor JMS	**Campo de MQMD utilizado para la transmisión**	**Cabecera**	**Establecido por**
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	Codificación		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 JMS en send () o publish ()
Estas notas están relacionadas con la correlación de campos JMS en send () o publish ().

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 898 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 898 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 JMS PERSISTENT es equivalente a MQPER_PERSISTENT
- El valor JMS NON_PERSISTENT 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 mensajes exclusivos asignados por WebSphere 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. WebSphere MQ messageId es un valor binario de 24 bytes, mientras que JMSMessageID es una serie. 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 posibilidad de especificar el MQMD de MQMD.MessageID, puede hacerlo con una de las extensiones JMS de WebSphere MQ descritas en "Lectura y grabación del descriptor de mensaje desde una aplicación WebSphere MQ classes for JMS" en la página 914.

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. Consulte también el tema "Utilización de una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker" en la página 942.

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. En MQRFH2 no se codifica ningún valor 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 JMS
Estas notas hacen referencia a la correlación de campos de propiedad JMS en mensajes de WebSphere 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 en mensajes IBM WebSphere MQ .

JMS_IBM_Report_ < nombre > a informe MQMD
Una aplicación JMS puede establecer las opciones de informe MQMD, utilizando las siguientes propiedades JMS_IBM_Report_XXX. El MQMD único se correlaciona con varias propiedades JMS_IBM_Report_XXX. La aplicación debe establecer el valor de estas propiedades en las constantes MQRO_ estándar de IBM WebSphere MQ (que se incluyen en com.ibm.mq.MQC). Por ejemplo, para solicitar confirmación de entrega con datos completos, la aplicación debe establecer JMS_IBM_Report_COD en el valor CMQC.MQRO_COD_WITH_FULL_DATA.

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

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ con campos JMS (mensajes de entrada)
Estas tablas muestran cómo se correlacionan los campos de propiedad y cabecera JMS en los campos MQMD y MQRFH2 al obtener () o recibir ().

La Tabla 116 en la página 842 muestra cómo se correlacionan los campos de cabecera JMS en los campos MQMD/MQRFH2 en el momento de obtener () o recibir (). Tabla 117 en la página 842 muestra cómo se correlacionan los campos de propiedad JMS en campos MQMD/MQRFH2 durante la hora get () o receive (). Tabla 118 en la página 843 muestra cómo se correlacionan las propiedades específicas del proveedor JMS.

Tabla 116. Correlación de campos de cabecera JMS de mensaje de entrada

Nombre de campo de cabecera JMS	Campo de MQMD recuperado de	Campo de MQRFH2 recuperado de
JMSDestination		jms.Dst o mqps.Top "1" en la página 842
JMSDeliveryMode	persistencia"2" en la página 842	jms.Dlv"2" en la página 842
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate"2" en la página 842 PutTime"2" en la página 842	jms.Tms"2" en la página 842
JMSCorrelationID	CorrelId"2" en la página 842	jms.Cid"2" en la página 842
JMSReplyTo	ColaRespuesta"2" en la página 842 GestorColasRespuesta"2" en la página 842	jms.Rto"2" en la página 842
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 serie que contiene el equivalente de juego de caracteres Java para el valor numérico CodedCharacterSetId .

Tabla 117. Correlación de propiedades para mensajes entrantes

Nombre de la propiedad JMS	Campo de MQMD recuperado de	Campo de MQRFH2 recuperado de
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId"1" en la página 842	jms.Gid"1" en la página 842
JMSXGroupSeq	NúmeroSecMsj"1" en la página 842	jms.Seq"1" en la página 842

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 118. Correlación de propiedades JMS específicas del proveedor de mensajes de entrada		
Nombre de la propiedad 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 843	Codificación	
JMS_IBM_Character_Set "1" en la página 843	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 JMS y una aplicación WebSphere MQ tradicional
En este tema se describe lo que sucede cuando una aplicación JMS intercambia mensajes con una aplicación WebSphere MQ tradicional que no puede procesar la cabecera MQRFH2 .

. Figura 129 en la página 844 muestra la correlación.

El administrador indica que la aplicación JMS se comunica con una aplicación WebSphere 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 tradicional de WebSphere MQ es la misma que la correlación de JMS con MQMD dirigida a una aplicación JMS. Si WebSphere MQ classes for JMS recibe un mensaje WebSphere MQ con el campo MQMD *Format* establecido en otro valor que no sea MQFMT_RFH2, se reciben datos de una aplicación no JMS. Si el formato es MQFMT_STRING, el mensaje se recibe como un mensaje de texto JMS. De lo contrario, se recibe como un mensaje de bytes JMS. Puesto que no hay ninguna MQRFH2, sólo se pueden restaurar las propiedades JMS que se transmiten en MQMD.

Si WebSphere MQ classes for JMS recibe un mensaje que no tiene una cabecera MQRFH2 , la propiedad TARGCLIENT del objeto Queue o Topic derivado 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.

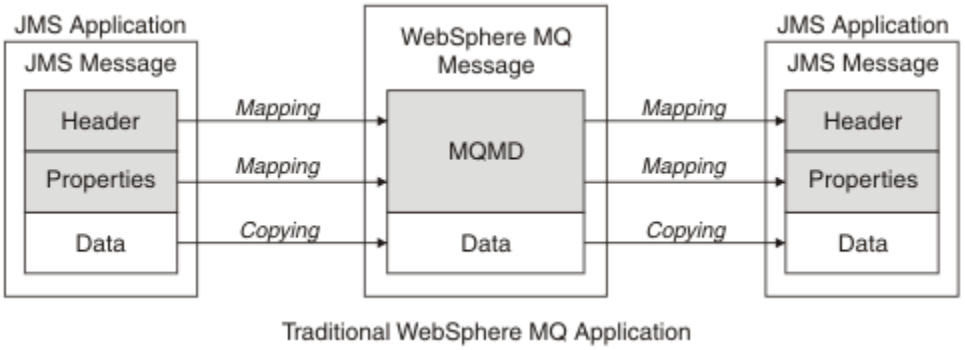


Figura 129. Cómo se transforman los mensajes JMS en mensajes WebSphere MQ sin cabecera MQRFH2

El cuerpo del mensaje JMS
Este tema contiene información sobre la codificación del propio cuerpo del mensaje. La codificación depende del tipo de mensaje JMS.

ObjectMessage
Un ObjectMessage es un objeto serializado por Java Runtime de la forma normal.

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). Sin embargo, es posible especificar cualquier otro juego de caracteres soportado por WebSphere MQ classes for JMS. Estos juegos de caracteres se utilizan principalmente cuando se envía un mensaje a una aplicación no 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 WebSphere MQ (o MQMD si no hay cabeceras). Para los mensajes JMS, la última cabecera suele ser MQRFH2.

BytesMessage
Un BytesMessage es, de forma predeterminada, una secuencia de bytes tal como se define en la especificación JMS 1.0.2 y en la documentación Java asociada.

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 S/390 en lugar de en formato IEEE).

Un mensaje entrante se interpreta utilizando la codificación numérica especificada en el mensaje. Esta especificación está en la última cabecera WebSphere MQ (o MQMD si no hay cabeceras). Para los mensajes JMS, la última cabecera suele ser 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. Se codifica en formato UTF8 de Java y empieza 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 del juego de caracteres en un mensaje entrante de WebSphere MQ no tiene ningún efecto en la interpretación de dicho mensaje como BytesMessagede JMS.

Es poco probable que las aplicaciones no Java reconozcan la codificación UTF8 de Java. Por lo tanto, para que una aplicación JMS envíe un BytesMessage que contenga 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 109 en la página 834. El tipo de datos predeterminado es `string`, por lo tanto, el atributo `dt="string"` se omite para los elementos de 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 WebSphere MQ classes for JMS anteriores a la versión 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>
```

La versión 5.3 y versiones posteriores de WebSphere MQ classes for JMS pueden interpretar cualquiera de los formatos, pero las versiones de WebSphere MQ classes for JMS anteriores a la versión 5.3 no pueden interpretar el formato actual.

Si una aplicación necesita enviar mensajes de correlación a otra aplicación que utiliza una versión de WebSphere MQ classes for JMS anterior a la versión 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 envían 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 109 en la página 834. El tipo de datos predeterminado es `string`, por lo tanto, el atributo `dt="string"` se omite para los elementos de 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 mensaje en JMS se realiza al enviar y recibir mensajes. WebSphere MQ realiza la mayor parte de la conversión de datos automáticamente. Convierte texto y datos numéricos al transferir un mensaje entre aplicaciones JMS. El texto se convierte al intercambiar un `JMSTextMessage` entre una aplicación JMS y una aplicación WebSphere 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 WebSphere 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 numéricos y de texto 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. WebSphere MQ gestiona la codificación de texto y números en mensajes JMS, excepto para `JMSObjectMessage`, consulte “JMSObjectMessage” en la página 853. Utiliza tres atributos de mensaje. Estos son `CodedCharacterSetId`, `Encoding` y `Format`.

Estos tres atributos de mensaje se almacenan normalmente en los campos de cabecera JMS, `MQRFH2`, de un mensaje JMS. Si el tipo de mensaje es un tipo de mensaje `MQ`, en lugar de `JMS` , los atributos se almacenan en el descriptor de mensaje, `MQMD`. Los atributos se utilizan para convertir los datos del mensaje JMS. Los datos de mensaje JMS se transfieren en la parte de datos de mensaje de un mensaje de WebSphere MQ .

Propiedades de mensaje de 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 se pueden enviar `JMSTextMessage` y `JMSBytesMessage` sin un `MQRFH2`. Si una propiedad JMS se almacena como una propiedad de mensaje WebSphere MQ en el descriptor de mensaje, `MQMD`, se convierte como parte de la conversión `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

Varios enfoques de conversión de datos están abiertos a los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si la aplicación sólo está intercambiando texto o sólo está intercambiando mensajes con otras aplicaciones JMS, normalmente no tiene en cuenta la conversión de datos. La conversión de datos se realiza automáticamente mediante WebSphere 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 las transferencias de mensajes JMS a JMS y el intercambio de mensajes de texto con programas IBM WebSphere MQ , IBM WebSphere MQ realiza automáticamente las conversiones necesarias. 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 847
2. “Conversión de datos de aplicación” en la página 847
3. “Conversión de datos de gestor de colas” en la página 848
4. “Conversión de datos de canal de mensajes” en la página 849

¿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 849 se describe el enfoque habitual de "el receptor lo hace bien". "El receptor es bueno" también se aplica a la conversión de datos JMS.

Conversión de datos de cliente JMS

Cliente JMS[4]la conversión de datos es la conversión de primitivos y objetos Java en bytes en un mensaje JMS a medida que se envía a un destino, y la conversión de nuevo, 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 119 en la página 850.

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 está siguiendo el patrón "el receptor es bueno" , o si está intercambiando mensajes entre aplicaciones JMS, la aplicación no necesita establecer `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 la propiedad de juego de caracteres de destino que se va a aplicar.

Para obtener los datos de un mensaje, una aplicación llama a los métodos de lectura u obtención de mensajes JMS. Los métodos hacen referencia a la página de códigos y a la codificación definidas en la cabecera de mensaje anterior para crear las primitivas y los objetos Java correctamente.

La conversión de datos de cliente JMS satisface las necesidades de la mayoría de 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 860.

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 en Figura 132 en la página 852

[4] "Cliente JMS" hace referencia a las clases WebSphere MQ para JMS que implementan la interfaz JMS, que se ejecuta en modalidad de cliente o de enlaces.

y Figura 133 en la página 852 . 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[5]. También puede enviar y recibir bytes utilizando `JMSStreamMessage`, `JMSMapMessage`o `JMSObjectMessage`.

No hay métodos Java para codificar y descodificar bytes que contengan 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 para la conversión de datos de aplicación es si un cliente JMS envía o recibe un mensaje formateado de una aplicación no 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 construye como `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 formas de convertir un mensaje formateado en un objeto Java que es fácil de utilizar.

1. Construya una clase Java correspondiente al registro, para encapsular la lectura y 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. Construya una clase Java correspondiente al registro ampliando 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 868

Conversión de datos de gestor de colas

En WebSphere MQ V7.0, el gestor de colas puede realizar la conversión de página 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 `GetMsgOpts` de `MQGET`; consulte Figura 131 en la página 852. Un gestor de colas realiza la conversión para un programa cliente JMS que recibe un mensaje, si la propiedad de destino `WMQ_RECEIVE_CONVERSION` se establece en `WMQ_RECEIVE_CONVERSION_QMGR`, el programa cliente JMS también puede establecer la propiedad de destino; consulte Figura 130 en la página 849.

Antes de V7.0, las conversiones siempre las realizaba el cliente JMS. La conversión de datos de cliente JMS está restringida a la conversión de secuencias de números y texto de tipo y longitud conocidos por el cliente JMS. No puede convertir estructuras de datos; consulte “Intercambio de un registro formateado con una aplicación no JMS” en la página 868. En V7.0, hasta el fixpack 7.0.1.5, si el gestor de colas puede realizar la conversión, entonces siempre será este quien la realice. A partir de 7.0.1.5 , el comportamiento de conversión predeterminado vuelve a ser el mismo que V6.0, y todas las conversiones las realiza el cliente JMS. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, puede establecer una nueva opción de destino, `WMQ_RECEIVE_CONVERSION`, para controlar dónde se realiza la conversión, y `WMQ_RECEIVE_CCSID`, para establecer la página de códigos de destino; consulte Figura 130 en la página 849.

[5] Una excepción: los datos escritos utilizando `writeUTF` empiezan con un campo de longitud de 2 bytes

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

O,

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figura 130. Habilitar la conversión de datos de gestor de colas

La principal ventaja de la conversión del gestor de colas se produce cuando se intercambian mensajes con aplicaciones no 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 mensajes predefinidos de WebSphere MQ , como por ejemplo una cabecera de puente CICS (`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. No es necesario que un cliente JMS emisor realice la conversión. Un programa de recepción no JMS se basa en la salida de conversión para asegurarse de que el mensaje se entrega en la página de códigos y la codificación necesarias. Con un cliente JMS de envío y un receptor no JMS, el ejemplo se aplica a IBM WebSphere MQ pre-and post-V7.0. Con IBM WebSphere MQ V7.0, también se puede llamar a la salida de conversión para un programa JMS receptor.

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 clase Java y utilizar su propia 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 868

Conversión de datos de canal de mensajes

WebSphere MQ Los canales remitente, servidor, clúster receptor y clúster emisor 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 método habitual en el diseño de la aplicación WebSphere MQ para la conversión de código es "el receptor hace que sea bueno". "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 orientación 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 que necesita el receptor, si el receptor no es un cliente JMS, puede eliminar la necesidad de que el destinatario no JMS realice la conversión. Si el destinatario es un cliente JMS, se va a convertir de nuevo, de todos modos, para decodificar los datos del mensaje y crear primitivos y objetos Java.

La diferencia entre las aplicaciones cliente JMS y las aplicaciones escritas en un lenguaje como C, es que Java debe realizar la 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 del mensaje, puede establecer el juego de caracteres y la codificación utilizados por WebSphere MQ para codificar números y texto en los mensajes. Normalmente, dejaría el juego de caracteres como `1208` y la codificación como `Native`.

WebSphere 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 valor predeterminado de `Charset` suele ser `windows-1252` en Windowsy `UTF-8` en UNIX. `windows-1252` es un juego de caracteres de un solo byte y `UTF-8` es un juego de caracteres de varios bytes.

Generalmente, deje el juego de caracteres de destino y las propiedades de codificación en sus valores predeterminados de `UTF-8` y `Native` al intercambiar mensajes con otras aplicaciones JMS. Si está intercambiando mensajes que contienen números o texto con una aplicación JMS, elija uno de los tipos de mensajes `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage`o `JMSObjectMessage` que se ajusten a su propósito. 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 las clases IBM WebSphere MQ para JMS añadan información adicional de cabecera y etiquetado 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 868 para ver ejemplos de conversión de mensajes intercambiados con aplicaciones no JMS.

Ejemplos

Tabla de tipos de mensajes y tipos de conversión

Tabla 119. Tipos de mensajes y tipos de conversión

	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

Tabla 119. 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

Llamada a la conversión de datos desde un programa C

```
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 131. Fragmento de código de amqsget0.c

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 132 en la página 852 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir un mensaje de serie de texto en bytes que contenga una mezcla de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 133 en la página 852. 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 132. 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 133. Recepción de una String de un JMSBytesMessage

Conceptos relacionados

Conversión y codificación de mensajes de cliente JMS
Se listan los métodos que utiliza 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 del gestor de colas siempre ha estado disponible para las aplicaciones no JMS que reciben mensajes de clientes JMS. Desde V7.0, los clientes JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, la conversión de datos de gestor de colas 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 formateado con una aplicación no JMS puede tener lugar con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensajes 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 mensajes JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagey JMSBytesMessage.

Conversión y tipos de mensajes 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 mensajes JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagey 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 sólo se puede intercambiar con otras aplicaciones JMS. Al elegir uno de los otros tipos de mensajes JMS, es posible intercambiar mensajes JMS con aplicaciones no JMS.

"Envío y recepción de un JMSObjectMessage" en la página 856 muestra un objeto String que se intercambia en un mensaje.

Una aplicación cliente JMS sólo puede recibir un JMSObjectMessage 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 texto Format 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 WebSphere MQ. Los campos CodedCharacterSetId y Format se establecen en el descriptor de mensaje, MQMD, o en los campos JMS de un 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 856.

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 856.

Un JMSTextMessage se puede enviar 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 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 del gestor de colas está habilitada, el texto lo convierte el gestor de colas, pero el cliente JMS debe seguir recibiéndolo como una matriz 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 mensajes 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 858. Los detalles se describen en “Conversión y codificación de mensajes de cliente JMS” en la página 860. Cada primitiva está etiquetada; consulte “El cuerpo del mensaje JMS” en la página 844.

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 138 en la página 857, 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 más detalles. El ejemplo “Envío de datos en un JMSStreamMessage y en un JMSMapMessage” en la página 857 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, el gestor de colas convierte los datos del mensaje antes de enviarlos al cliente 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 JMS sólo puede recibir un JMSMapMessage o JMSStreamMessage en un mensaje que tenga un cuerpo de estilo JMS, y desde 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 858. Los detalles se describen en "Conversión y tipos de mensajes JMS" en la página 853.

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 decodificar texto en bytes utilizando la clase Charset . Transfiere los bytes entre la JVM y el mensaje sin que las clases WebSphere MQ para JMS realicen ninguna conversión; consulte "Envío y recepción de texto en un JMSBytesMessage" en la página 857.

Un JMSBytesMessage enviado a una aplicación MQ normalmente se envía 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 envía un JMSBytesMessage con un cuerpo de estilo MQ , puede recibir el mensaje de un destino que defina un MQ o un estilo de cuerpo de mensaje JMS. Si envía un JMSBytesMessage con un cuerpo de estilo JMS, debe recibir el mensaje de un destino que defina un estilo de cuerpo de 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.

Si un mensaje tiene un MQ o un estilo de cuerpo JMS, la forma en que se recibe no se ve afectada por el valor 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. El código basado en el ejemplo, "Lectura y escritura de mensajes con DataInputStream y DataOutputStream" en la página 857, es necesario para utilizar 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 134. 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 135. 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 136. 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 137. 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 138. Envío de datos en un JMSStreamMessage y en un JMSMapMessage

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 139 en la página 857 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir un mensaje de serie de texto en bytes que contenga una mezcla de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 140 en la página 857. 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 139. 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 140. Recepción de una String de un JMSBytesMessage

Lectura y escritura de mensajes con DataInputStream y DataOutputStream

El código en Figura 141 en la página 858 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 141. 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 al escribir 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 142 en la página 858 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 142. 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 codificado numérico.

Tabla de tipos de mensajes y tipos de conversión

Tabla 120. Tipos de mensajes y tipos de conversión

	Tipo de conversión			
Tipo de mensaje	**Texto**	**Numérico**	**Otro**	**Ninguna**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tabla 120. 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

Varios enfoques de conversión de datos están abiertos a los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si la aplicación sólo está intercambiando texto o sólo está intercambiando mensajes con otras aplicaciones JMS, normalmente no tiene en cuenta la conversión de datos. La conversión de datos se realiza automáticamente mediante WebSphere MQ.

Conversión y codificación de mensajes de cliente JMS

Se listan los métodos que utiliza 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 del gestor de colas siempre ha estado disponible para las aplicaciones no JMS que reciben mensajes de clientes JMS. Desde V7.0, los clientes JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, la conversión de datos de gestor de colas 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 formateado con una aplicación no JMS puede tener lugar con o sin llamar a una salida de conversión de datos.

Conversión y codificación de mensajes de cliente JMS
Se listan los métodos que utiliza 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 las primitivas u objetos Java se leen o se escriben en y desde 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 los datos se leen o se escriben en un mensaje JMS. El texto se convierte a y desde la representación Unicode interna de 16 bits[6]para el juego de caracteres utilizado para el texto de los mensajes. Los datos numéricos se convierten a y los tipos numéricos primitivos Java a la codificación definida para el mensaje. Si se realiza la conversión, y qué tipo de conversión se realiza, depende del tipo de mensaje JMS y de la operación de lectura o grabación.

Tabla 121 en la página 860 categoriza los métodos de lectura y escritura para distintos tipos de mensajes JMS por el tipo de conversión realizado. Los tipos de conversión se describen en el texto que sigue a la tabla.

Tabla 121. 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`

[6] Alguna representación Unicode requiere más de 16 bits. Consulte una referencia de Java SE.

Tabla 121. 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[7].

Los fragmentos de código de "Envío y recepción de un JMSTextmessage" en la página 864 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 147 en la página 864 recibe el mensaje que se ha enviado al destino. El texto del mensaje se convierte de la página de códigos IBM037 de nuevo a Unicode.

Nota: Una forma sencilla de comprobar que el texto se convierte al juego de caracteres codificado 37 es utilizar WebSphere MQ Explorer. Examine la cola y muestre las propiedades del mensaje antes de que se recupere.

[7] Al recibir un mensaje JMS_IBM_CHARacter_SET es un nombre de página de códigos de Java Charset .

Contraste el fragmento de código de Figura 146 en la página 864 con el fragmento de código incorrecto de Figura 143 en la página 862. En el fragmento de código incorrecto, la serie de texto se convierte dos veces, una vez por parte de la aplicación, y otra vez por parte de WebSphere MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figura 143. 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. Al recibir, 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 148 en la página 864 muestra una aplicación que codifica un número en el formato de destino y lo envía en un `JMSStreamMessage`. Compare el ejemplo de Figura 148 en la página 864 con el ejemplo de Figura 149 en la página 864. La diferencia es que `JMS_IBM_ENCODING` se debe enviar en un `JMSBytesMessage`.

Nota: Una forma sencilla de comprobar que el número se ha codificado correctamente es utilizar WebSphere 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 865.

WebSphere MQ no realiza ninguna conversión de página de códigos en el cliente porque el mensaje es un `JMSBytesMessage`y porque 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 receptor pueda seguir el convenio de convertir cualquier matriz de bytes que represente texto en el mensaje en series o caracteres utilizando la propiedad `JMS_IBM_CHARACTER_SET` en el 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. Las clases WebSphere MQ para Java utilizan `JMS_IBM_CHARacter_SET` para establecer el campo `CodedCharacterSetId` en las propiedades JMS en `MQRFH2`, o en el descriptor de mensaje, `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(wMQConstants.JMS_IBM_CHARACTER_SET, codePage);[8]
```

Si una matriz de bytes se escribe en `JMSStringMessage` o `JMSMapMessage`, WebSphere MQ classes for JMS no realiza la conversión de datos, ya que los bytes se escriben como datos hexadecimales, no como texto en `JMSStringMessage` y `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 144 en la página 863 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 UNIX, `UTF-8`. El intercambio entre Windows y UNIX requiere que seleccione 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 144. Escritura de bytes que representan una serie en un `JMSStreamMessage` utilizando el juego de caracteres de destino

Ejemplos

[8] `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 145. 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 146. 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 147. 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 148. 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 149. Envío de un número utilizando la codificación de destino en un JMSBytesMessage

Envío y recepción de texto en un JMSBytesMessage

El código de Figura 150 en la página 865 envía una serie en un BytesMessage. Por simplicidad, el ejemplo envía una serie individual, para la que JMSTextMessage es más adecuado. Para recibir un mensaje de serie de texto en bytes que contenga una mezcla de tipos, debe conocer la longitud de la serie en bytes, denominada *TEXT_LENGTH* en Figura 151 en la página 865. 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 150. 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 151. Recepción de una String de un JMSBytesMessage

Conceptos relacionados

Enfoques de conversión de mensajes JMS

Varios enfoques de conversión de datos están abiertos a los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si la aplicación sólo está intercambiando texto o sólo está intercambiando mensajes con otras aplicaciones JMS, normalmente no tiene en cuenta la conversión de datos. La conversión de datos se realiza automáticamente mediante WebSphere MQ.

Conversión de datos de gestor de colas

La conversión de datos del gestor de colas siempre ha estado disponible para las aplicaciones no JMS que reciben mensajes de clientes JMS. Desde V7.0, los clientes JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, la conversión de datos de gestor de colas 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 formateado con una aplicación no JMS puede tener lugar con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensajes 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 mensajes JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagey JMSBytesMessage.

Conversión de datos de gestor de colas

La conversión de datos del gestor de colas siempre ha estado disponible para las aplicaciones no JMS que reciben mensajes de clientes JMS. Desde V7.0, los clientes JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, la conversión de datos de gestor de colas 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 no JMS, la prestación de conversión siempre ha estado disponible estableciendo la opción GetMessage, GMO_CONVERT. La capacidad de conversión del gestor de colas no ha estado disponible para una aplicación JMS que recibe un mensaje hasta V7.0.

Puede utilizar la conversión del gestor de colas, antes de V7.0, con una aplicación cliente JMS que envíe un mensaje. El cliente JMS crea un registro formateado, establece los atributos CodedCharacterSetId, Encodingy Format correspondientes a los datos colocados en el mensaje. Una aplicación receptora no JMS lee el mensaje utilizando GMO_CONVERTy hace que se llame a una salida de conversión de datos escrita por el usuario. La salida de conversión de datos es una biblioteca compartida que tiene el nombre establecido en el campo Format.

Desde V7.0, el gestor de colas puede convertir mensajes que se envían a clientes JMS. De 7.0.0.0 a 7.0.1.4 inclusive, siempre se llama a la conversión del gestor de colas para los clientes JMS. A partir de la versión 7.0.1.5, o la versión 7.0.1.4 con APAR IC72897 aplicado, la conversión de gestor de colas se controla estableciendo la propiedad de destino, WMQ_RECEIVE_CONVERSION, en WMQ_RECEIVE_CONVERSION_QMGR o en WMQ_RECEIVE_CONVERSION_CLIENT_MSG. WMQ_RECEIVE_CONVERSION_CLIENT_MSG es el valor predeterminado, que coincide con el comportamiento de WebSphere MQ V6.0, que no daba soporte a la conversión de datos del gestor de colas para clientes JMS. 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 152. Habilitar la conversión de datos de gestor de colas

La conversión de datos del 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 en 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 153. 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 de texto de mensaje en UTF-8.

El código JMS es equivalente al texto en negrita en el código C en Figura 154 en la página 867,

```
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 154. 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 WebSphere 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
Varios enfoques de conversión de datos están abiertos a los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si la aplicación sólo está intercambiando texto o sólo está intercambiando mensajes con otras aplicaciones JMS, normalmente no tiene en cuenta la conversión de datos. La conversión de datos se realiza automáticamente mediante WebSphere MQ.

Conversión y codificación de mensajes de cliente JMS
Se listan los métodos que utiliza 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 425
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 formateado con una aplicación no JMS puede tener lugar con o sin llamar a una salida de conversión de datos.

Referencia relacionada

Conversión y tipos de mensajes 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 mensajes JMS, JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagey 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 formateado con una aplicación no JMS puede tener lugar 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 un `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 formato de mensajes JMS intercambiados con otros clientes JMS. Siempre que el tipo de mensaje sea `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessageo` `JMSObjectMessage`, WebSphere MQ se ocupa de los detalles del formato del mensaje. WebSphere MQ trata las diferencias en las páginas de códigos y la codificación numérica en distintas plataformas.

Puede utilizar estos tipos de mensajes para intercambiar mensajes con aplicaciones no JMS. Para ello, debe comprender cómo construyen estos mensajes las clases de WebSphere MQ para JMS. Es posible que pueda modificar la aplicación no JMS para interpretar los mensajes; consulte "Correlación de mensajes JMS con mensajes WebSphere MQ" en la página 829.

Una ventaja de utilizar uno de estos tipos de mensajes es que la programación de cliente JMS no depende del tipo de aplicación con la que está intercambiando 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 pueda tratar con formatos de mensaje existentes. A menudo, los mensajes existentes tienen un formato fijo y contienen una mezcla de datos, texto y números sin formato. Utilice los pasos de esta tarea y el cliente JMS de ejemplo en "Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage" en la página 871, 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 WebSphere MQ predefinidas.

   Para manejar cabeceras de WebSphere MQ predefinidas, consulte Manejo de cabeceras de mensajes de WebSphere MQ.

   Figura 155 en la página 869 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 871, indique la salida de conversión de datos `MYRECORD`.
3. Escriba clases Java para encapsular el diseño de registro y enviar y recibir el registro. 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 871.
   - 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 lo hace bien” en la página 849.

5. Configure el destino para intercambiar mensajes de tipo MQ, sin una cabecera MQRFH2 JMS.

   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 871 se sustituye la propiedad de cuerpo de mensaje del destino, garantizando así que se envía un mensaje de estilo MQ.

6. Probar 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 156 en la página 870. Con las modificaciones, `amqsgetc0.c` recibe un mensaje enviado por el cliente JMS de ejemplo, `TryMyRecord.java`; consulte “Escritura de clases para encapsular un diseño de registro en un archivo JMSBytesMessage” en la página 871.
   - El programa de exploración de WebSphere MQ de ejemplo, `amqsbcg0.c`, es útil para inspeccionar el contenido de la cabecera de 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 155. 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 156. Modificación de amqsget0.c

Conceptos relacionados

Enfoques de conversión de mensajes JMS

Varios enfoques de conversión de datos están abiertos a los diseñadores de aplicaciones JMS. Estos enfoques no son exclusivos; es probable que algunas aplicaciones utilicen una combinación de estos enfoques. Si la aplicación sólo está intercambiando texto o sólo está intercambiando mensajes con otras aplicaciones JMS, normalmente no tiene en cuenta la conversión de datos. La conversión de datos se realiza automáticamente mediante WebSphere MQ.

Conversión y codificación de mensajes de cliente JMS

Se listan los métodos que utiliza 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 del gestor de colas siempre ha estado disponible para las aplicaciones no JMS que reciben mensajes de clientes JMS. Desde V7.0, los clientes JMS que reciben mensajes también utilizan la conversión de datos del gestor de colas. A partir de 7.0.1.5, o 7.0.1.4 con APAR IC72897, la conversión de datos de gestor de colas es opcional.

Referencia relacionada

Conversión y tipos de mensajes 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 mensajes JMS, `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessagey` `JMSBytesMessage`.

Utilidad para crear código de salida de conversión

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, puede crear algunas clases 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 tipos de datos mixtos con programas no JMS. El proveedor JMS no ha insertado datos adicionales en el cuerpo del mensaje. Por lo tanto, es la mejor opción de tipo de mensaje a utilizar si un programa cliente JMS interactúa con un programa IBM WebSphere 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 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 Java y JMS. No se proporcionan instrucciones sobre cómo configurar el entorno de desarrollo 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 V7.0.
3. El ejemplo se ha creado utilizando la perspectiva Java del entorno de trabajo Eclipse . Se necesita JRE 6.0 o superior. Puede utilizar la perspectiva Java en IBM WebSphere MQ Explorer para desarrollar y ejecutar las clases Java. De forma alternativa, utilice su propio entorno de desarrollo Java.
4. El uso de IBM WebSphere 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 WebSphere 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 WebSphere 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 878 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
Clase abstracta que encapsula la conexión JMS, el destino y la sesión en una sola 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 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 WebSphere 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 WebSphere 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 WebSphere MQ utilizan identificadores de juegos de caracteres codificados. El código de ejemplo utiliza juegos de caracteres.

d. `int` se serializa en `MQLONG` mediante IBM WebSphere 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 el captador 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 adelante, 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());`" llama a "`(((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 las clases IBM WebSphere MQ para JMS no lo utilizan al escribir un `JMSBytesMessage`.

   Llamadas de "`messageCharset = myProducer.getCharset();`"

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

   Obtiene el juego de caracteres 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 cliente MQI de IBM WebSphere MQ . 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 IBM WebSphere MQ en unspecified, si debe alterar temporalmente un valor de JMS.

Nota:

Las clases IBM WebSphere MQ para JMS escriben el formato, la codificación y el identificador de juego de caracteres del mensaje en el descriptor de mensaje, MQMD, o en la cabecera JMS, MQRFH2. Depende de si el mensaje tiene un cuerpo de estilo de IBM WebSphere 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 JMS_IBM_* , las propiedades JMS_IBM_MQMD_* no controlan cómo las clases IBM WebSphere MQ para JMS construyen un mensaje JMS. Es posible 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 javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. 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; }Â .
```

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

Resultados

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 mensajes en el juego de caracteres codificado 37 y *no* 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 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 156 en la página 870 en “Intercambio de un registro formateado con una aplicación no JMS” en la página 868.
  - 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 mensajes en el juego de caracteres codificado 37 y *no* utilizando 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 WebSphere MQy ejecute el programa principal, `TryMyRecord`; consulte Figura 157 en la página 878.

1. Configure IBM WebSphere MQ y JMS para ejecutar el ejemplo. Las instrucciones son para ejecutar el ejemplo en Windows.

   1. Crear un gestor de colas

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

2. Crear una cola

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

3. Crear un directorio JNDI

```
cd c:\
md JNDI-Directory
```

4. Conmutar al directorio bin de JMS
El programa de administración JMS debe ejecutarse desde aquí. La vía de acceso es *MQ_INSTALLATION_PATH*\java\bin.

5. Cree las siguientes definiciones 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
```

6. 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 WebSphere MQ Explorer.
3. Ejecute `TryMyRecord`.

Clases utilizadas para ejecutar el ejemplo

Las clases listadas en las figuras Figura 157 en la página 878 a Figura 162 en la página 882 también están disponibles en un archivo comprimido; descargue jm25529_.zip o jm25529_.tar.gz.

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

Figura 157. TryMyRecord

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

Figura 158. RECORD

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }Â .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

Figura 159. MyRecord

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                  == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
       }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

Figura 160. EndPoint

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

Figura 161. MyProducer

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Figura 162. MyConsumer

Creación y configuración de fábricas de conexiones y destinos en una aplicación WebSphere MQ classes for JMS

Una aplicación WebSphere MQ para JMS 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 JMS de IBM o utilizando las extensiones JMS de WebSphere MQ . Una aplicación también puede utilizar las extensiones JMS de IBM o las extensiones JMS de WebSphere MQ para establecer las propiedades de fábricas de conexiones y destinos.

Las fábricas de conexiones y los destinos son puntos de partida en el flujo de la lógica de una aplicación JMS. 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 JMS de WebSphere MQ o WebSphere 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 JMS de IBM o las extensiones JMS de WebSphere MQ .

Utilización de las extensiones JMS de IBM

Una aplicación puede utilizar las extensiones JMS de IBM para crear fábricas de conexiones y destinos dinámicamente en 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 JMS de WebSphere MQ

Una aplicación también puede utilizar las extensiones JMS de WebSphere MQ para crear fábricas de conexiones y destinos dinámicamente en 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.

Utilización de JNDI para recuperar objetos administrados en una aplicación JMS

Para recuperar objetos administrados de un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación JMS 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. El administrador puede utilizar la herramienta de administración JMS de WebSphere MQ o WebSphere MQ Explorer para crear y mantener objetos administrados en un espacio de nombres JNDI. Para obtener información sobre cómo utilizar la herramienta de administración JMS de WebSphere MQ , consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953. Para obtener información sobre cómo utilizar WebSphere MQ Explorer, consulte la ayuda proporcionada con WebSphere MQ Explorer. Sin embargo, un servidor de aplicaciones normalmente proporciona su propio repositorio para objetos administrados y sus propias herramientas para la creación y mantenimiento de 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:

```
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

Utilización de las extensiones JMS de IBM

WebSphere MQ classes for JMS contiene un conjunto de extensiones para la API JMS denominadas extensiones JMS de IBM . Una aplicación puede utilizar estas extensiones para crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución y para establecer las propiedades de las clases WebSphere MQ para objetos JMS. Las extensiones se pueden utilizar con cualquier proveedor de mensajería.

Las extensiones JMS de IBM son un conjunto de interfaces y clases en los paquetes siguientes:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Los paquetes se pueden encontrar en `com.ibm.mqjms.jar` que se encuentra en `<MQ_Install_Dir>/java/lib`.

Estas extensiones proporcionan las funciones siguientes:

- Un mecanismo basado en fábrica para crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, en lugar de recuperarlos como objetos administrados desde un espacio de nombres JNDI (Java Naming and Directory Interface)
- Un conjunto de métodos para establecer las propiedades de las clases WebSphere MQ para objetos JMS
- 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 las clases WebSphere MQ para JMS

Con respecto a la creación dinámica de fábricas de conexiones y destinos en tiempo de ejecución, y a la configuración y obtención de sus propiedades, las extensiones JMS de IBM proporcionan un conjunto alternativo de interfaces a las extensiones JMS de WebSphere MQ . Sin embargo, mientras que las extensiones JMS de WebSphere MQ son específicas del proveedor de mensajería WebSphere MQ , las extensiones JMS de IBM no son específicas de WebSphere MQ y se pueden utilizar con cualquier proveedor de mensajería dentro de la arquitectura en capas descrita en “Una arquitectura en capas” en la página 816.

La interfaz com.ibm.msg.client.wmq.WMQConstants contiene las definiciones de constantes, que una aplicación puede utilizar al establecer las propiedades de las clases WebSphere MQ para objetos JMS

utilizando las extensiones JMS IBM . La interfaz contiene constantes para el proveedor de mensajería WebSphere MQ y constantes JMS que son independientes de cualquier proveedor de mensajería.

En los siguientes ejemplos de código se da por supuesto que se han ejecutado estas sentencias de importación:

```
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

Para que una aplicación pueda crear fábricas de conexiones y destinos utilizando las extensiones JMS de IBM , primero debe crear un objeto de fábrica JmsFactory. Para crear un objeto JmsFactoryFactory, la aplicación llama al método getInstance() de la clase JmsFactoryFactory, tal como se muestra en el ejemplo siguiente:

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

El parámetro en la llamada getInstance() es una constante que identifica el proveedor de mensajería de WebSphere MQ como el 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 las clases WebSphere MQ para objetos JMS” en la página 886.

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 WebSphere 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 WebSphere 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 de WebSphere MQ denominada Q2 propiedad del gestor de colas QM2y 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 898. Para conocer un procedimiento alternativo para establecer las propiedades de un objeto JmsQueue, consulte “Establecimiento de las propiedades de las clases WebSphere MQ para objetos JMS” en la página 886.

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 898. Para conocer un procedimiento alternativo para establecer las propiedades de un objeto JmsTopic, consulte “Establecimiento de las propiedades de las clases WebSphere MQ para objetos JMS” en la página 886.

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 las clases WebSphere MQ para objetos JMS

Para establecer las propiedades de las clases de WebSphere MQ para objetos JMS utilizando las extensiones JMS de IBM , una aplicación utiliza los métodos de la interfaz com.ibm.msg.client.JmsPropertyContext .

Para cada tipo de datos Java, la interfaz de contexto JmsPropertycontiene 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 del paquete com.ibm.mq.jms también heredan los métodos de la interfaz JmsPropertyContext. 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 WebSphere MQ para JMS, 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, WebSphere MQ classes for JMS emite una excepción JMSException. Si una aplicación intenta obtener una propiedad que no se ha establecido, el comportamiento es el que se describe en la especificación JMS. WebSphere MQ classes for JMS emite una excepción NumberFormatpara los tipos de datos primitivos y devuelve un valor nulo para los tipos de datos referenciados.

Además de las propiedades predefinidas de un objeto WebSphere MQ para JMS, una aplicación puede establecer sus propias propiedades. Estas propiedades definidas por la aplicación son ignoradas por las clases de WebSphere MQ para JMS.

Para obtener más información sobre las propiedades de las clases WebSphere MQ para objetos JMS, consulte Propiedades de objetos IBM WebSphere MQ classes for JMS.

El código siguiente es un ejemplo de cómo establecer propiedades utilizando las extensiones JMS de IBM . 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 plataformas z/OS no dan soporte al establecimiento de nombres de aplicación y, por lo tanto, este valor se ignora.

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 WebSphere MQ para JMS, 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 WebSphere MQ classes for JMS se muestran en Tabla 122 en la página 888.

Tabla 122. Conversiones soportadas de un tipo de datos a otro	
Tipo de datos de origen	**Tipos de datos de destino soportados**
boolean	Cadena
byte	int, long, short, String
carácter	Cadena
doble	Cadena
flotante	double, String
int	long, String
largo	Cadena
corto	int, long, String
Cadena	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 se puede convertir a 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, WebSphere MQ classes for JMS emite una excepción de excepción NumberFormat.
- Si una aplicación intenta una conversión que no está soportada, WebSphere MQ classes for JMS emite una 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 está en el formato correcto, WebSphere MQ classes for JMS emite 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:

  [*blancos*] [*signo*]*dígitos*[*e_char*[*e_sign*]*e_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.

  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 está en el formato correcto, WebSphere MQ classes for JMS emite 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 a la representación de serie del valor como un número decimal, no a 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 las clases WebSphere MQ para objetos JMS” en la página 886. 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 JmsPropertypara establecer y obtener las propiedades de las clases de WebSphere MQ para objetos JMS, la aplicación puede especificar los nombres y valores de las propiedades de cualquiera de las maneras 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.

Utilización de los nombres y valores de propiedad reconocidos por la herramienta de administración JMS de WebSphere MQ

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, WebSphere MQ classes for JMS almacena nombres y valores de propiedad como los valores literales definidos en la interfaz com.ibm.msg.client.wmq.WMQConstants . 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, WebSphere MQ classes for JMS tiene 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, las clases WebSphere MQ para JMS 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 excepciones, en archivos de rastreo o en el registro de WebSphere MQ para JMS siempre están en 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 WebSphere MQ de clases para JMS.

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 JMS de WebSphere MQ

WebSphere MQ classes for JMS contiene un conjunto de extensiones para la API JMS denominadas extensiones JMS de WebSphere MQ . 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.

WebSphere 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 JMS y contienen las extensiones JMS de WebSphere MQ . 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.*;
```

Una aplicación puede utilizar las extensiones JMS de WebSphere MQ para realizar las funciones siguientes:

- Crear fábricas de conexiones y destinos dinámicamente en tiempo de ejecución, en lugar de recuperarlos como objetos administrados desde 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 892.

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.

Para una conexión en tiempo real con un intermediario, una aplicación puede utilizar el código siguiente:

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

Este código presupone que el intermediario se ejecuta en un sistema con el nombre de host HOST2 y está a la escucha en el número de puerto 1507.

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 WebSphere MQ clases para objetos 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 WebSphere 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 WebSphere 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 WebSphere 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, WebSphere MQ debe estar configurado para que, cuando la aplicación envíe un mensaje a este destino, Websphere MQ pueda direccionar el mensaje del 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 WebSphere 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 de WebSphere 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 898. 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 893.

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 898. 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 893.

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 WebSphere MQ classes for JMS.

Creación de una conexión en 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 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 JMS, IBM WebSphere MQ classes for JMS crea un descriptor 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();
```

Creación de una sesión en una aplicación JMS

Para crear una sesión, 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 JMS, las clases IBM WebSphere MQ classes for JMS crean un descriptor de conexión (Hconn) e inician 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.

Sesiones con transacción en aplicaciones JMS

Las aplicaciones JMS pueden ejecutar transacciones locales creando primero una sesión de transacción. 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 sesiones 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_RECONOCER y DUPS_OK_RECONOCER, JMS no da soporte a que una aplicación emita una excepción no manejada 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 JMS

En lugar de recuperar destinos como objetos administrados desde un espacio de nombres JNDI (Java Naming and Directory Interface), una aplicación JMS puede utilizar una sesión para crear destinos

dinámicamente en tiempo de ejecución. Una aplicación puede utilizar un identificador uniforme de recursos (URI) para identificar una cola o un tema de WebSphere MQ y, opcionalmente, para 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 WebSphere 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 WebSphere 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 de WebSphere 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, WebSphere MQ debe estar configurado para que, cuando la aplicación envíe un mensaje a este destino, Websphere MQ pueda direccionar el mensaje del 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 898.

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, WebSphere 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 898.

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, WebSphere MQ classes for JMS crea un tema con un nombre que empieza por 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 WebSphere 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, WebSphere MQ debe estar configurado para que, cuando una aplicación envíe un mensaje a la cola, Websphere MQ pueda direccionar el mensaje del 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 WebSphere MQ .

La cola puede ser una cola local, una cola alias o una definición de cola remota.

Para ver las reglas para crear nombres de cola, consulte Reglas para nombrar objetos de IBM WebSphere MQ .

topicName
El nombre del tema.

Para las reglas para crear nombres de tema, consulte Reglas para designar objetos IBM WebSphere 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 Utilización de series de tema.

***propertyName1*, *propertyName2*, ...**
Los nombres de las propiedades del objeto Queue o Topic creado por la aplicación. La Tabla 123 en la página 899 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 123 en la página 899 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 123 en la página 899 lista los nombres de propiedad válidos que se pueden utilizar en los URI de cola y de tema. Aunque la herramienta de administración JMS de WebSphere MQ utiliza constantes simbólicas para los valores de las propiedades, los URI no pueden contener constantes simbólicas.

Tabla 123. Nombres de propiedad y valores válidos para utilizar en los URI de cola y de tema

Nombre de propiedad	**Descripción**	**Valores válidos**
CCSID	Cómo se representan los datos de tipo carácter en el cuerpo de un mensaje cuando WebSphere MQ classes for JMS reenvía el mensaje al destino	• Cualquier identificador de juego de caracteres codificado soportado por WebSphere MQ.
codificación	Cómo se representan los datos numéricos en el cuerpo de un mensaje cuando WebSphere MQ classes for JMS reenvía el mensaje al destino	• Cualquier valor válido para el campo *Codificación* en un descriptor de mensaje WebSphere 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.

Tabla 123. Nombres de propiedad y valores válidos para utilizar en los URI de cola y de tema (continuación)

Nombre de propiedad	Descripción	Valores válidos
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. Asociado con cada valor está el valor correspondiente de la propiedad MULTICAST tal como se utiliza en la herramienta de administración JMS de WebSphere MQ . Para obtener una descripción de la propiedad MULTICAST y sus valores válidos, consulte Propiedades de objetos IBM WebSphere MQ classes for JMS. • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED
persistencia	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 especifica en el atributo *DefPersistence* de la cola o tema WebSphere MQ . • 1 - No persistente • 2 - Persistente • 3-Equivalente al valor HIGH para la propiedad PERSISTENCE tal como se utiliza en la herramienta de administración JMS de WebSphere MQ . Para obtener una explicación de este valor, consulte “Mensajes persistentes JMS” en la página 923.
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 especifica en el atributo *DefPriority* de la cola o tema WebSphere MQ . • Un entero comprendido dentro del rango 0-9 que especifica la prioridad de los mensajes enviados al destino.

Tabla 123. Nombres de propiedad y valores válidos para utilizar en los URI de cola y de tema (continuación)

Nombre de propiedad	Descripción	Valores válidos
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 WebSphere 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 WebSphere 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 JMS

Para que una aplicación 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" en la página 883
- "Utilización de las extensiones JMS de IBM" en la página 884
- "Utilización de las extensiones JMS de WebSphere MQ" en la página 891
- "Creación de destinos en una aplicación JMS" en la página 896

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 de datos (Stream)
- Texto

Cada tipo de cuerpo de mensaje tiene su propia interfaz JMS, que es una subinterfaz de la interfaz Message, y un método en la interfaz Session 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 JMS" en la página 825.

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, WebSphere MQ classes for JMS reenvía el mensaje y devuelve el control a la aplicación sin determinar si el gestor de colas ha recibido el mensaje de forma segura. Esto a veces se denomina *transferencia asíncrona*. Para obtener más información, consulte "Colocación asíncrona de mensajes en clases IBM WebSphere MQ para JMS" en la página 939.

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" en la página 883
- "Utilización de las extensiones JMS de IBM" en la página 884
- "Utilización de las extensiones JMS de WebSphere MQ" en la página 891
- "Creación de destinos en una aplicación JMS" en la página 896

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 extensión en WebSphere MQ classes for JMS, un suscriptor de tema no duradero también recibe publicaciones retenidas, pero no cuando se utiliza una conexión en tiempo real con un intermediario.

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 extensión en WebSphere 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 JMS y las propiedades de mensaje. Para obtener información sobre cómo crear un identificador de mensajes, consulte “Selectores de mensajes en JMS” en la página 826.

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:

```
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)
- 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 WebSphere MQ classes for JMS” en la página 908.

V 7.5.0.8 Recuperación de datos de usuario de suscripción

Si los mensajes que una aplicación de IBM WebSphere 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.

A partir de Version 7.5.0, Fix Pack 8, cuando se consume un mensaje de una cola que contiene una cabecera RFH2 con la carpeta MQPS, el valor asociado con la clave Sud, si existe, se añade como una propiedad String al objeto Mensaje JMS devuelto a la aplicación IBM WebSphere MQ classes for JMS . Para habilitar la recuperación de esta propiedad desde el mensaje, se puede utilizar la constante JMS_IBM_SUBSCRIPTION_USER_DATA en la interfaz JmsConstants con el método javax.jms.Message.getStringProperty(java.lang.String) para obtener los datos de usuario de suscripción.

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 IBM WebSphere MQ classes for JMS puede llamar a:

```
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 830

Tareas relacionadas

Definir una suscripción administrativa

Referencia relacionada

DEFINE SUB

Interfaz JmsConstants

Cierre de una aplicación WebSphere MQ para JMS

Es importante que una aplicación WebSphere MQ para JMS cierre determinados objetos JMS de forma explícita 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 todas las clases WebSphere MQ para JMS y recursos WebSphere MQ de forma puntual, especialmente si una aplicación crea muchos objetos JMS de corta duración 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 finalizadores en una aplicación para cerrar objetos 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.

Manejo de mensajes no entregables en IBM WebSphere MQ classes for JMS

Un mensaje dañado es un mensaje que no puede procesar una aplicación MDB receptora. Si se encuentra un mensaje con formato incorrecto, los objetos JMS MessageConsumer y ConnectionConsumer pueden volver a ponerlo en cola de acuerdo con dos propiedades de cola, BOQNAME y BOTHRESH.

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 restitiuya 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 se denominan *mensajes no entregables*. El objeto MessageConsumer de JMS detecta mensajes con formato incorrecto y los redirecciona a un destino alternativo.

El gestor de colas de IBM WebSphere MQ mantiene un registro del número de veces que se ha retirado 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. Consulte “Eliminación de mensajes de la cola en ASF” en la página 948 para obtener más detalles.

Existe una diferencia en la forma en que MessageConsumers y ConnectionConsumers vuelven a poner cola los mensajes no entregables. ConnectionConsumers pueden poner en cola los mensajes no entregables sin afectar la entrega de los mensajes. El proceso de puesta en cola tiene lugar fuera de cualquier unidad de trabajo asociada con la entrega real de mensajes al código de la aplicación. Esto es posible debido a la naturaleza multihebra del funcionamiento de ConnectionConsumer.

En cambio los MessageConsumers tienen una sola hebra por debajo del nivel de sesión, y la puesta en cola de los mensajes no entregables tiene lugar dentro de la unidad de trabajo actual. Esto no afecta al funcionamiento de la aplicación, pero cuando se ponen en cola mensajes no entregables bajo una sesión transaccional o Client_acknowledge, la puesta en cola no se confirma hasta que la unidad de trabajo actual es confirmada por el código de aplicación o, si procede, el código del contenedor de aplicaciones.

Los objetos ConnectionConsumer de JMS manejan mensajes con formato incorrecto de la misma forma y utilizando 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 WebSphere 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 del proveedor de mensajería de IBM WebSphere MQ, es la cola gestionada del tema que se crea a partir de la cola modelo.

- Para la mensajería de publicación/suscripción en la modalidad de migración del proveedor de mensajería IBM WebSphere MQ, esta es la cola CCSUB definida en el objeto TopicConnectionFactory, o la cola CCDSUB definida en el objeto Topic.

IBM WebSphere MQ classes for JMS consulta los atributos BackoutThreshold y BackoutRequeueQName de la cola. Por lo tanto, debe otorgar acceso de consulta sobre la cola al usuario que ejecuta la aplicación.

V 7.5.0.9 Si la cola de destino es una cola de clúster, las autorizaciones necesarias dependen de la versión del IBM WebSphere MQ classes for JMS que se está utilizando:

- Cuando se utiliza IBM WebSphere MQ classes for JMS for Version 7.5.0, Fix Pack 9 más un arreglo temporal para el APAR IT26482, es necesario consultar el acceso.
- Para todas las demás versiones, otorgue inquire, examine y obtenga acceso.

Para establecer los atributos BackoutThreshold y BackoutRequeueQName, emita el mandato MQSC siguiente:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

Si el atributo BackoutThreshold se establece en un valor distinto de cero, para evitar un comportamiento inesperado, establezca el atributo BackoutRequeueQName en un nombre de cola válido.

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 WebSphere MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Para modificar estos valores, utilice:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value) BOQNAME(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 restituciones alcanza el valor umbral, pero el mensaje no puede pasar a la cola de retirada, el mensaje se envía a la cola de mensajes no entregados o 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. Para obtener información más detallada, consulte el apartado “Eliminación de mensajes de la cola en ASF” en la página 948.

Cuando se vuelve a poner un mensaje en la cola de reposición en cola para restitución, algunos de los valores de campo del descriptor de mensaje (MQMD) del mensaje cambian. Consulte MQMD - Descriptor de mensaje para obtener detalles sobre el formato del MQMD.

Los siguientes campos de MQMD cambian el valor cuando el mensaje entra en la cola de retirada.

- PutDate se actualiza a la fecha en la que va a la cola de reposición en cola para restitución.
- PutTime se actualiza en el momento en que va a la cola de reposición en cola para restitución.
- El recuento de restituciones se restablece en cero.
- La caducidad del mensaje se actualiza para reflejar la caducidad restante en el momento en que la aplicación JMS recibió el mensaje original.

Los valores de los campos siguientes siguen siendo los mismos cuando el mensaje entra en la cola de retirada:

- StructId
- Versión
- Informe
- MessageType
- Comentarios
- Codificación
- CodedCharSetId

- MsgId
- CorrelId
- ReplyToQ
- GestorColasRespuesta
- Formato
- Persistence
- Priority

Excepciones en IBM WebSphere MQ classes for JMS

Una aplicación IBM WebSphere MQ classes for JMS debe ser capaz de manejar excepciones generadas por llamadas de API JMS o entregadas a un manejador de excepciones.

IBM WebSphere MQ classes for JMS notifica problemas en tiempo de ejecución emitiendo excepciones. JMSException es la clase raíz para excepciones generadas por métodos JMS, y la captura de excepciones JMSException proporciona una forma genérica de manejar todas las excepciones relacionadas con JMS.

Cada excepción JMSException encapsula la información siguiente:

- Un mensaje de excepción específico del proveedor, que una aplicación obtiene llamando al método Throwable.getMessage().
- Un código de error específico del proveedor, que una aplicación obtiene llamando al método JMSException.getErrorCode().
- Una excepción enlazada. Una excepción generada por una llamada de API JMS suele ser el resultado de un problema de nivel inferior, que se notifica mediante otra excepción enlazada a esta excepción. Una aplicación obtiene una excepción enlazada llamando al método JMSException.getLinkedException() o Throwable.getCause().

La mayoría de excepciones emitidas por IBM WebSphere 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, que una aplicación obtiene llamando al método JmsExceptionDetail.getExplanation().
- Una respuesta del usuario recomendada para la excepción, que una aplicación obtiene llamando al método JmsExceptionDetail.getUserAction().
- Las claves para las inserciones de mensajes en el mensaje de excepción. Una aplicación obtiene un repetidor para todas las claves llamando al método JmsExceptionDetail.getKeys().
- Las inserciones de mensajes en el mensaje de excepción. Por ejemplo, una inserción de mensaje podría ser el nombre de la cola que ha provocado la excepción y podría ser útil para que una aplicación pueda acceder a ese nombre. Una aplicación obtiene la inserción de mensaje correspondiente a una clave especificada llamando al método JmsExceptionDetail.getValue().

Todos los métodos en la interfaz JmsExceptionDetail podrían devolver un valor nulo si no hay detalles disponibles.

Por ejemplo, si una aplicación intenta crear un productor de mensajes para una cola IBM WebSphere 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 WebSphere MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

La excepción generada, com.ibm.msg.client.jms.DetailedInvalidDestinationException, es una subclase de javax.jms.InvalidDestinationException e implementa la interfaz com.ibm.msg.client.jms.JmsExceptionDetail.

Excepciones enlazadas

Una excepción enlazada proporciona más información sobre un problema de tiempo de ejecución. Por tanto, para cada excepción JMSException que se genera, una aplicación debe comprobar 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(). Para una excepción JMSException, el método getLinkedException() realmente delega 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
                   |
                   +--->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, es emitida por la capa común en IBM WebSphere MQ classes for JMS.
- La excepción siguiente, una instancia de com.ibm.mq.MQException, es emitida por el proveedor de mensajería de IBM WebSphere MQ.
- La siguiente excepción, una instancia de com.ibm.mq.jmqi.JmqiException, la emite la interfaz Java común a la MQI.
- La excepción final, una instancia de java.net.ConnectionException, la genera la biblioteca de clases Java.

Para obtener más información sobre la arquitectura en capas de IBM WebSphere MQ classes for JMS, consulte “Clases IBM WebSphere MQ para la arquitectura JMS” en la página 816.

Mediante un código similar al siguiente, una aplicación puede ejecutar un proceso iterativo sobre esta cadena para extraer toda la información pertinente:

```
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 una aplicación siempre debe comprobar el tipo de cada excepción de una cadena porque el tipo de excepción puede variar y las excepciones de tipos diferentes encapsulan información diferente.

Obtención de información específica de IBM WebSphere MQ sobre un problema

Las instancias de com.ibm.mq.MQException y com.ibm.mq.jmqi.JmqiException encapsulan información específica de IBM WebSphere MQ sobre un problema.

Una excepción MQException encapsula la información siguiente:

- Un código de terminación, que una aplicación obtiene llamando al método getCompCode().
- Un código de razón, que una aplicación obtiene llamando al método getReason().

Una excepción JmqiException también encapsula un código de terminación y un código de razón. Además, sin embargo, una excepción JmqiException encapsula la información en un mensaje AMQ*nnnn* o CSQ*nnnn* , si hay uno asociado con la excepción. Al llamar a los métodos adecuados de la excepción, una aplicación puede obtener los diversos componentes de este mensaje, como por ejemplo, la gravedad, la explicación y la respuesta del usuario.

Para ver ejemplos de cómo utilizar los métodos mencionados en este apartado, consulte el código de ejemplo en “Excepciones enlazadas” en la página 911.

Actualización desde versiones anteriores de IBM WebSphere MQ classes for JMS

En comparación con las versiones anteriores de IBM WebSphere MQ classes for JMS, la mayoría de los códigos de error y mensajes de excepción han cambiado en la versión 7. La razón de estos cambios es que IBM WebSphere MQ classes for JMS ahora tiene una arquitectura en capas y se generan excepciones desde distintas capas del código.

Por ejemplo, si una aplicación intenta conectar con un gestor de colas que no existe, una versión anterior de IBM WebSphere MQ classes for JMS emitía una excepción JMSException con la información siguiente:

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

Esta excepción contenía una excepción enlazada MQException con la información siguiente:

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

En comparación en las mismas circunstancias, la versión 7 de IBM WebSphere MQ classes for JMS genera una excepción JMSException con la siguiente información:

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

Esta excepción contiene una excepción enlazada MQException con la información siguiente:

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

Si la aplicación analiza o prueba mensajes de excepción devueltos por el método Throwable.getMessage(), o códigos de error devueltos por el método JMSException.getErrorCode(), y está actualizando desde un release anterior a la versión 7, es probable que sea necesario modificar la aplicación para poder utilizar la versión 7 de IBM WebSphere MQ classes for JMS.

Escuchas de excepciones

Una aplicación puede registrar un escucha de excepción con un objeto Connection. Posteriormente, si se produce un problema que hace que la conexión sea inutilizable, IBM WebSphere MQ classes for JMS entrega una excepción al escucha de excepción invocando el método onException(). A continuación, la aplicación tiene la oportunidad de volver a establecer la conexión.

V 7.5.0.8 El APAR IT14820, incluido a partir de IBM WebSphere MQ Version 7.5.0, Fixpack 8, corrigió un defecto en el que no se invocaba el ExceptionListener JMS de una aplicación para excepciones de interrupción sin conexión (por ejemplo MQRC_GET_INHIBITED) aun cuando la propiedad ASYNC_EXCEPTIONS de la fábrica de conexiones JMS utilizada por la aplicación, estuviera establecida en ASYNC_EXCEPTIONS_ALL. Este era el valor predeterminado antes de Version 7.5.0, Fix Pack 8.

V 7.5.0.8 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 WebSphere MQ classes for JMS son coherentes con la especificación JMS, el valor predeterminado de la propiedad ConnectionFactory de JMS ASYNC_EXCEPTIONS se ha cambiado a ASYNC_EXCEPTIONS_CONNECTIONBROKEN para IBM WebSphere MQ classes for JMS. Como resultado, de forma predeterminada, solo las excepciones correspondientes a los códigos de error de conexión interrumpida se entregan a ExceptionListener JMS de una aplicación.

V 7.5.0.8 A partir de Version 7.5.0, Fix Pack 8, IBM WebSphere MQ classes for JMS se ha actualizado también de modo que las excepciones JMSExceptions relacionadas con errores de interrupción de no conexión, que se producen durante la entrega de mensajes a consumidores de mensajes asíncronos, todavía se entregan a un ExceptionListener registrado cuando la ConnectionFactory de JMS utilizada por la aplicación tiene la propiedad ASYNC_EXCEPTIONS establecida en el valor ASYNC_EXCEPTIONS_ALL.

V 7.5.0.8 Para obtener más información sobre qué ha cambiado para los escuchas de excepciones para Version 7.5.0, Fix Pack 8 y por qué se han realizado los cambios desde releases anteriores, consulte JMS: Exception listener changes in Version 7.5.

Para cualquier otro tipo de problema, la llamada de API JMS actual emite una excepción JMSException.

Si una aplicación no registra un escucha de excepciones con un objeto Connection, las excepciones que se habrían entregado al escucha de excepciones se graban en el registro de IBM WebSphere MQ classes for JMS .

Referencia relacionada

ASYNCEXCEPTION

Registro de errores en clases de WebSphere MQ para JMS

La información sobre los problemas de tiempo de ejecución que pueden requerir una acción correctiva por parte del usuario se graba en el registro de WebSphere MQ classes for JMS.

Por ejemplo, si una aplicación intenta establecer una propiedad de una fábrica de conexiones, pero no se reconoce el nombre de la propiedad, WebSphere MQ classes for JMS escribe información sobre el problema en su registro.

De forma predeterminada, el archivo de registro se denomina mqjms.log y reside en el directorio de trabajo actual. Sin embargo, puede cambiar el nombre y la ubicación del archivo de registro estableciendo la propiedad com.ibm.msg.client.commonservices.log.outputName en el archivo de configuración de clases de WebSphere MQ para JMS. Para obtener información sobre las clases WebSphere MQ para el archivo de configuración JMS, consulte “El archivo de configuración IBM WebSphere MQ classes for JMS” en la página 741y para obtener más detalles sobre los valores válidos para la propiedad com.ibm.msg.client.commonservices.log.outputName , consulte “Registro yIBM WebSphere MQ classes for JMS” en la página 813.

Tecnología de soporte de primera anomalía (FFST) en clases WebSphere MQ para JMS

Si se produce un error interno grave en las clases WebSphere MQ para JMS, se genera información de tecnología de soporte de primera anomalía (FFST).

La información de FFST se graba en un archivo denominado JMSC*nnnn*.FDC, donde *nnnn* es un número de cuatro dígitos. Este archivo se encuentra en un directorio denominado FFDC, que es un subdirectorio del directorio en el que se graba la salida de rastreo. De forma predeterminada, la salida de rastreo se graba en el directorio de trabajo actual, pero puede redirigir la salida de rastreo a un directorio diferente estableciendo la propiedad **com.ibm.msg.client.commonservices.trace.outputName** en el archivo de configuración WebSphere MQ classes for JMS. Para obtener información sobre el archivo de configuración de WebSphere MQ classes for JMS, consulte “El archivo de configuración IBM WebSphere MQ classes for JMS” en la página 741.

Si el rastreo está habilitado cuando se genera información FFST , la información FFST también se graba en el archivo de rastreo. Para obtener más información sobre el rastreo de programas JMS, consulte Rastreo de aplicaciones IBM WebSphere MQ classes for JMS.

Para suprimir la producción de archivos FFDC, establezca la propiedad **com.ibm.msg.client.commonservices.ffst.suppress**, como se indica a continuación:

0
: Salida de todos los archivos FFDC (valor predeterminado).

-1
: Generar sólo los primeros archivos FFDC de un tipo determinado.

entero
: Suprimir todos los archivos FFDC excepto los que son un múltiplo de este número.

Acceso a las características de WebSphere MQ desde una aplicación WebSphere MQ classes for JMS

WebSphere MQ classes for JMS proporciona recursos para aprovechar una serie de características de WebSphere MQ.

Atención: Estas características están fuera de la especificación JMS o, en determinados casos, violan la especificación JMS. Si los utiliza, es poco probable que la aplicación sea compatible con otros proveedores JMS. Las características que no cumplen con la especificación JMS se etiquetan con un aviso de atención.

Lectura y grabación del descriptor de mensaje desde una aplicación WebSphere 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 WebSphere MQ requieren que se establezcan valores específicos en el MQMD de los mensajes que se les envían. Las clases WebSphere MQ para JMS proporcionan atributos de mensaje que permiten a las aplicaciones JMS establecer campos MQMD y, por lo tanto, habilitar las aplicaciones JMS para "controlar" las aplicaciones WebSphere 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 JMS” en la página 917.

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 de objeto de destino JMS
Dos propiedades del objeto Destination controlan el acceso al MQMD desde JMS, y una tercera controla el contexto del mensaje.

Tabla 124. Nombres y descripciones de propiedades

Propiedad	**Formato abreviado**	**Descripción**
WMQ_MQMD_WRITE_ENABLED	MDW	Si una aplicación JMS puede establecer los valores de los campos MQMD
WMQ_MQMD_READ_ENABLED	MDR	Si una aplicación JMS puede extraer los valores de los campos MQMD

Tabla 124. Nombres y descripciones de propiedades (continuación)

Propiedad	Formato abreviado	Descripción
WMQ_MQMD_MESSAGE_ CONTEXT	MDCTX	Qué nivel de contexto de mensaje debe establecer la aplicación JMS. La aplicación se debe ejecutar con la autorización de contexto apropiada para que esta propiedad entre en vigor

Tabla 125. 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** • True	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** • True	setMQMDReadEnabled

Tabla 125. 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 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 126. Nombres, descripciones y tipos de propiedades

Propiedad	Descripción	Tipo Java	Enlace a la descripción completa
JMS_IBM_MQMD_Report	Opciones para mensajes de informe	Entero	Report
JMS_IBM_MQMD_MsgType	Tipo de mensaje	Entero	MsgType
JMS_IBM_MQMD_Expiry	Duración del mensaje	Entero	Expiry
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	Cadena	Format
JMS_IBM_MQMD_Priority [1]	Prioridad de mensaje	Entero	Priority
JMS_IBM_MQMD_Persistence	Persistencia de los mensajes	Entero	Persistence
JMS_IBM_MQMD_MsgId [2]	Identificador del 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	Cadena	ReplyToQ

Tabla 126. Nombres, descripciones y tipos de propiedades (continuación)

Propiedad	Descripción	Tipo Java	Enlace a la descripción completa
JMS_IBM_MQMD_ReplyToQMgr	Nombre del gestor de colas de respuestas	Cadena	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Identificador de usuario	Cadena	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Señal de contabilidad	Object (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	Datos de aplicación relacionados con la identidad	Cadena	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	Cadena	PutApplName
JMS_IBM_MQMD_PutDate	Fecha cuando se colocó el mensaje	Cadena	PutDate
JMS_IBM_MQMD_PutTime	Hora cuando se colocó el mensaje	Cadena	PutTime
JMS_IBM_MQMD_ApplOriginData	Datos de aplicación relacionados con el origen	Cadena	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	Offset
JMS_IBM_MQMD_MsgFlags	Distintivos de mensajes	Entero	MsgFlags
JMS_IBM_MQMD_OriginalLength	Longitud del mensaje original	Entero	OriginalLength

Tabla 126. Nombres, descripciones y tipos de propiedades (continuación)

Propiedad	Descripción	Tipo Java	Enlace a la descripción completa

1. **Atención:** Si asigna un valor a JMS_IBM_MQMD_Priority que no está dentro del rango 0-9, esto viola la especificación JMS.
2. **Atención:** La especificación JMS indica que el ID de mensaje debe ser establecido por el proveedor JMS y que se debe ser exclusivo o nulo. Si asigna un valor a JMS_IBM_MQMD_MsgId, este valor se copia en JMSMessageID. Por lo tanto, no lo establece el proveedor JMS y puede que no sea exclusivo: esto infringe la especificación JMS.
3. **Atención:** Si asigna un valor a JMS_IBM_MQMD_CorrelId que empieza por la serie 'ID:', esto viola la especificación JMS.
4. **Atención:** El uso de las propiedades de matriz de bytes en un mensaje infringe la especificación JMS.

Acceso a datos de IBM WebSphere MQ Message desde una aplicación utilizando clases WebSphere MQ para JMS

Puede acceder a los datos completos del mensaje WebSphere MQ dentro de una aplicación utilizando clases IBM WebSphere MQ para 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 WebSphere 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` se establece en `WMQ_MESSAGE_BODY_JMS` o `WMQ_MESSAGE_BODY_UNSPECIFIED`, el cuerpo del mensaje se devuelve sin la cabecera `MQRFH2` de JMS 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 a un gestor de colas de WebSphere MQ V6 , o si ha establecido `PROVIDERVERSION` en 6, las funciones no están 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` se establece en `WMQ_MESSAGE_BODY_JMS`, WebSphere MQ classes for JMS genera automáticamente una cabecera `MQRFH2` basada en los valores de las propiedades y los campos de cabecera de `JMSMessage` .

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 en `WMQ_MESSAGE_BODY_UNSPECIFIED`, WebSphere MQ classes for JMS envía una cabecera `MQRFH2` , a menos que `WMQ_TARGET_CLIENT` se establezca en `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 WebSphere MQ . Utilice las propiedades siguientes:

- JMS_IBM_Format o JMS_IBM_MQMD_Format: esta propiedad especifica el formato de la cabecera WebSphere MQ o carga útil de aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera MQ de Websphere anterior.
- JMS_IBM_Character_Set o JMS_IBM_MQMD_CodedCharSetId: esta propiedad especifica el CCSID de la cabecera o carga útil de aplicación WebSphere MQ 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 WebSphere MQ o carga útil de 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 JMS de IBM WebSphere MQ .
2. Las propiedades JMS_IBM_MQMD_* solo se configuran en la MQMD. Las propiedades JMS_IBM_* se establecen en MQMD sólo si el mensaje no tiene una cabecera JMS MQRFH2 . De lo contrario, se establecen en la cabecera RFH2 de JMS.
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 163 en la página 921 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 "ëÈÊ'>...??>?"
```

Figura 163. Datos de mensaje y MQMD codificados de forma incoherente

Cualquiera de los fragmentos de código de Figura 164 en la página 922 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 164. Envío de un mensaje con un cuerpo de mensaje MQ.

Recepción de un mensaje

Si WMQ_MESSAGE_BODY se establece en WMQ_MESSAGE_BODY_JMS, el tipo y el cuerpo del mensaje JMS de entrada están determinados por el contenido del mensaje de 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 se establece 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 se establece en WMQ_MESSAGE_BODY_UNSPECIFIED, el valor predeterminado, IBM WebSphere MQ classes for JMS lo establece en 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 WebSphere MQ o carga útil de aplicación que inicia el cuerpo del mensaje JMS si no hay ninguna cabecera MQ de Websphere anterior.
- JMS_IBM_Character_Set o JMS_IBM_MQMD_CodedCharSetId: esta propiedad especifica el CCSID de la cabecera o carga útil de aplicación WebSphere MQ 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 WebSphere MQ o carga útil de 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 JMS procesa la MQRFH2 de un mensaje WebSphere MQ como parte de la carga útil del mensaje (es decir, como parte del cuerpo del mensaje JMS).

Tabla 127. Nombres y descripciones de propiedades		
Propiedad	**Formato abreviado**	**Descripción**
WMQ_MESSAGE_BODY	MBODY	Indica si una aplicación JMS procesa la MQRFH2 de un mensaje WebSphere MQ como parte de la carga útil del mensaje (es decir, como parte del cuerpo del mensaje JMS).

Tabla 128. 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** Al enviar, WebSphere MQ classes for JMS genera o no genera e incluye una cabecera MQRFH2 , en función del valor de WMQ_TARGET_CLIENT. Al recibir, actúa como JMS de valor. • JMS Al enviar, WebSphere MQ classes for JMS genera automáticamente una cabecera MQRFH2 y la incluye en el mensaje WebSphere MQ . Al recibir, WebSphere MQ classes for JMS establece las propiedades del mensaje JMS de acuerdo con los valores de la MQRFH2 (si está presente); no presenta la MQRFH2 como parte del cuerpo del mensaje JMS. • MQ Al enviar, WebSphere MQ classes for JMS no genera una MQRFH2. Al recibir, WebSphere MQ classes for JMS presenta MQRFH2 como parte del cuerpo del mensaje JMS.	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodySty le

Mensajes persistentes JMS

WebSphere Las clases MQ para aplicaciones JMS pueden utilizar el atributo de cola **NonPersistentMessageClass** para proporcionar un mejor rendimiento para los mensajes persistentes JMS, a expensas de cierta fiabilidad.

Una cola de WebSphere MQ tiene 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 el atributo para una cola local utilizando el mandato WebSphere MQ Script (MQSC), DEFINE QLOCAL, con cualquiera de los parámetros siguientes:

NPMCLASS(NORMAL)
Los mensajes no persistentes de la cola se descartan cuando se reinicia el gestor de colas. Este 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.

En este tema se describe cómo las clases WebSphere MQ para aplicaciones JMS pueden utilizar este atributo de cola para proporcionar un mejor rendimiento para los mensajes persistentes JMS.

La propiedad PERSISTENCE de un objeto Queue o Topic puede tener el valor HIGH. Puede utilizar la herramienta de administración JMS de WebSphere MQ 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 JMS o un mensaje no persistente JMS a un destino donde la propiedad PERSISTENCE tiene el valor HIGH, y la cola subyacente de WebSphere MQ se establece en NPMCLASS (HIGH), el mensaje se coloca en la cola como un mensaje no persistente de WebSphere 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 JMS se coloca en la cola como un mensaje persistente WebSphere MQ y un mensaje no persistente JMS se coloca en la cola como un mensaje no persistente WebSphere MQ .

Si un mensaje persistente JMS se coloca en una cola como un mensaje no persistente de WebSphere MQ , y desea asegurarse de que el mensaje no se descarta después de una conclusión inactiva o inmediata de un gestor de colas, todas las colas a través de las cuales se puede direccionar 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, WebSphere MQ classes for JMS emite una excepción InvalidDestinationsi una aplicación intenta crear un consumidor de mensajes para un destino donde la propiedad PERSISTENCE tiene el valor HIGH y la cola subyacente de WebSphere 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 JMS se recibe como un mensaje persistente JMS, y un mensaje enviado como un mensaje no persistente JMS se recibe como un mensaje no persistente 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, WebSphere MQ classes for JMS emite una llamada MQINQ para determinar si NPMCLASS (HIGH) está establecido en la cola subyacente de WebSphere MQ . Por consiguiente, la aplicación debe tener la autorización para consultar la cola. Además, WebSphere 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 NPMCLASS en la cola subyacente mientras la aplicación sigue utilizando el destino, WebSphere MQ classes for JMS no observa el nuevo valor.

Al permitir que los mensajes persistentes JMS se coloquen en colas de WebSphere MQ como mensajes no persistentes de WebSphere MQ , está obteniendo rendimiento a expensas de cierta fiabilidad. Si necesita la máxima fiabilidad para los mensajes persistentes JMS, no envíe los mensajes a un destino donde la propiedad PERSISTENCE tenga el valor HIGH.

La capa 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. Si desea utilizar colas

temporales para aceptar mensajes persistentes, debe utilizar SYSTEM.JMS.TEMPQ.MODEL, o cambie la cola modelo por una cola alternativa de su elección.

Utilización de SSL (Secure Sockets Layer) con clases WebSphere MQ para JMS

Las clases WebSphere MQ para aplicaciones JMS pueden utilizar el cifrado SSL. Para ello necesitan un proveedor JSSE.

Las clases WebSphere MQ para conexiones JMS que utilizan TRANSPORT (CLIENT) dan soporte al cifrado SSL (Secure Sockets Layer). SSL proporciona cifrado de comunicaciones, autenticación e integridad de mensajes. Se suele utilizar para proteger las comunicaciones entre dos interlocutores cualesquiera en Internet o dentro de una intranet.

WebSphere MQ classes for JMS utiliza JSSE (Java Secure Socket Extension) para manejar el cifrado SSL y, por lo tanto, requiere 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 WebSphere MQ classes for JMS utiliza una tabla de definición de canal de cliente (CCDT) para conectarse a un gestor de colas, consulte “Utilización de una tabla de definiciones de canal de cliente con IBM WebSphere MQ classes for JMS” en la página 934.

Propiedad de objeto SSLCIPHERSUITE
Establezca SSLCIPHERSUITE para habilitar el cifrado SSL en un objeto ConnectionFactory .

Para habilitar el cifrado SSL 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. “SSL CipherSpecs y CipherSuites en JMS” en la página 928 contiene una tabla que correlaciona las CipherSpecs soportadas por WebSphere MQ con sus CipherSuites equivalentes tal como las conoce JSSE. Para obtener más información sobre CipherSpecs y CipherSuites con WebSphere MQ, consulte Seguridad.

Por ejemplo, para configurar un objeto ConnectionFactory que se puede utilizar para crear una conexión a través de un canal MQI habilitado para SSL con una CipherSpec de RC4_MD5_EXPORT, emita el mandato siguiente a JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

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 WebSphere MQ classes for JMS” en la página 934.

propiedad de objeto SSLFIPSREQUIRED
Si necesita una conexión para utilizar una CipherSuite soportada por el proveedor IBM Java JSSE FIPS (IBMJSSEFIPS), establezca la propiedad SSLFIPSREQUIRED de la fábrica de conexiones en YES.

El valor predeterminado de esta propiedad es NO, lo que significa que una conexión puede utilizar cualquier CipherSuite soportada por WebSphere 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 UNIX, Linux y Windows

propiedad de objeto SSLPEERNAME

Utilice SSLPEERNAME para especificar un patrón de nombre distinguido, para asegurarse de que la aplicación JMS se conecta al gestor de colas correcto.

Una aplicación JMS puede asegurarse de que se conecta al gestor de colas correcto especificando un patrón de nombre distinguido (DN). 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 siguiente mandato JMSAdmin establece un objeto ConnectionFactory para que el gestor de colas se identifique a sí mismo con un nombre común que empiece con los caracteres `QMGR.`, y con al menos dos nombres de unidad organizativa, 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 CRL en Java 2 v1.4 o posterior. El siguiente ejemplo JMSAdmin indica a JMS que 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 Java Software Development Kit (SDK) sea 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 único punto de anomalía, JMS permite que se proporcionen varios servidores LDAP proporcionando una lista de servidores LDAP delimitados por el carácter de 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 intenta cada uno a su vez hasta que encuentra un servidor con el que puede verificar correctamente 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 la información de control enviada y recibida por las clases de WebSphere MQ para JMS.

Por ejemplo, para configurar un objeto ConnectionFactory que se puede utilizar para crear una conexión a través de un canal MQI habilitado para SSL con una clave secreta que se renegocia después de que hayan fluido 4 MB de datos, emita el mandato siguiente 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 SSL para una aplicación, cree una SSLSocketFactory y configure JMS para utilizarla.

Es posible que desee personalizar otros aspectos de la conexión SSL 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. Después de obtener un objeto SSLSocketFactory adecuado, utilice el método MQConnectionFactory.setSSLSocketFactory () para configurar JMS para utilizar 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 imitar el servidor LDAP, engañando a una aplicación JMS para que se conecte al servidor incorrecto sin darse cuenta. 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 JSSE, o cambia la ubicación del almacén de claves o del archivo de almacén de confianza, las clases WebSphere MQ para aplicaciones JMS que se están ejecutando en ese momento no recogen 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.

SSL CipherSpecs y CipherSuites en JMS
CipherSpecs soportadas por WebSphere MQ y sus CipherSuitesequivalentes.

La Tabla 129 en la página 929 lista las CipherSpecs soportadas por WebSphere MQ y sus CipherSuitesequivalentes. Si la propiedad ConnectionFactory SSLFIPSREQUIRED se establece en NO, una aplicación WebSphere MQ classes for JMS puede conectarse a un gestor de colas si se especifica alguna CipherSpec soportada en el extremo del servidor del canal MQI y se especifica la CipherSuite equivalente en el extremo del cliente. Si SSLFIPSREQUIRED se establece en YES, la combinación de CipherSpec y CipherSuite determina si la aplicación puede conectarse al gestor de colas.

En el extremo del servidor de un canal MQI, el nombre de una CipherSpec se puede especificar como el valor del parámetro SSLCIPH en un mandato DEFINE CHANNEL CHLTYPE (SVRCONN). En el extremo del cliente de un canal MQI, el nombre de una CipherSuite se puede especificar de las maneras siguientes:

- Una aplicación puede llamar al método setSSLCipherSuite () de un objeto ConnectionFactory .

- Utilizando la herramienta de administración JMS de WebSphere MQ , puede establecer la propiedad SSLCIPHERSUITE de un objeto ConnectionFactory .

Tabla 129. CipherSpecs soportadas por WebSphere MQ y sus CipherSuites equivalentes

CipherSpec	**CipherSuite equivalente**	**¿Es posible la conexión si SFIPS [1] está establecido en YES?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	No
NULL_SHA	SSL_RSA_WITH_NULL_SHA	No
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	No
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	No
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	No
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	No
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	No
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	No
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	No
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	No
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	No [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	Sí [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	Sí [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	Sí [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8] [9]	SSL_RSA_WITH_DES_CBC_SHA	No[3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Sí
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	No[4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	No[6]

Notas:

1. Cuando se utiliza la herramienta de administración JMS de WebSphere MQ , SFIPS es el nombre abreviado de la propiedad ConnectionFactory SSLFIPSREQUIRED.
2. Esta CipherSpec no tiene ninguna CipherSuiteequivalente.
3. Esta CipherSpec estaba certificada con FIPS 140-2 antes del 19th de mayo de 2007.
4. Esta CipherSpec estaba certificada con FIPS 140-2 antes del 19th de mayo de 2007. El nombre FIPS_WITH_DES_CBC_SHA es histórico y refleja el hecho de que esta CipherSpec anteriormente era compatible con FIPS (pero ya no lo es). Esta CipherSpec está en desuso y su uso no se recomienda.
5. Estas CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) no se pueden utilizar para proteger una conexión desde WebSphere MQ Explorer a un gestor de colas a menos que se apliquen los archivos de políticas no restringidas adecuados al JRE utilizado por el explorador.

   Consulte Información de seguridad para obtener más información sobre los archivos de políticas.

6. El nombre FIPS_WITH_3DES_EDE_CBC_SHA es histórico y refleja el hecho de que esta CipherSpec anteriormente era compatible con FIPS (pero ya no lo es). Esta CipherSpec está en desuso y su uso no se recomienda.
7. Estas CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) necesitan IBM JREs 6.0 SR13 FP2 , 7.0 SR4 FP2 o posterior.
8. Estas CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) pueden utilizar SSLv3 o TLS. De forma predeterminada, cuando FIPS no está habilitado, se utiliza SSLv3 . Para utilizar TLS, establezca la propiedad del sistema Java **com.ibm.mq.cfg.preferTLS** en `true`.
9. 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.

Información relacionada

Especificación de que sólo se utilizan CipherSpecs certificadas por FIPS en el tiempo de ejecución del cliente MQI

Federal Information Processing Standards (FIPS) para UNIX, Linux y Windows

Escritura de salidas de canal en Java para clases de WebSphere MQ para JMS

Las salidas de canal se crean definiendo clases Java que implementan interfaces especificadas.

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 las clases de WebSphere MQ para JMS deben 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 del servidor de una conexión, el cifrado SSL se realiza *después* de que se llame a cualquier salida de canal. De forma similar, cuando se reciben datos desde el extremo del servidor de una conexión, el descifrado SSL se realiza *antes* de que se llame a cualquier salida de canal.

En las versiones de WebSphere MQ classes for JMS anteriores a la versión 7.0, las salidas de canal se implementaban utilizando las interfaces MQSendExit, MQReceiveExity MQSecurityExit. Puede seguir utilizando estas interfaces, pero se prefieren las nuevas interfaces para obtener un rendimiento y funcionamiento mejores.

Configuración de IBM WebSphere MQ classes for JMS para utilizar salidas de canal

Una aplicación de IBM WebSphere 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 WebSphere MQ para la comunicación entrante.
- La propiedad **sendexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM WebSphere 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 WebSphere MQ para la comunicación entrante.
- La propiedad **receiveexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM WebSphere 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 WebSphere MQ para la comunicación entrante.
- La propiedad **securityexit** en un objeto ConnectionFactory utilizado por el adaptador de recursos IBM WebSphere MQ para la comunicación de salida.

Para MQConnectionFactories, puede establecer las propiedades **SENDEXIT**, **RECEXIT** y **SECEXIT** utilizando la herramienta de administración de IBM WebSphere MQ JMS o IBM WebSphere 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 WebSphere MQ.
2. 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 WebSphere MQ, tal como se muestra en la Tabla 130 en la página 932. 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 130. El directorio de salidas de IBM WebSphere MQ

Plataforma	**Directorio**
UNIX and Linux	`/var/mqm/exits` (salidas de canal de 32 bits) `/var/mqm/exits64` (salidas de canal de 64 bits)

Tabla 130. El directorio de salidas de IBM WebSphere 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 WebSphere MQ durante la instalación. El directorio predeterminado es `C:\Program Files\IBM\WebSphere MQ`.

Nota: Si existe una salida de canal en más de una ubicación, IBM WebSphere 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 WebSphere 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 WebSphere 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 políticas utilizado por el entorno de ejecución de Java en el que se está ejecutando la aplicación debe tener los permisos para cargar una clase de salida de canal. Para obtener información sobre cómo hacer esto, consulte Ejecución de aplicaciones de IBM MQ Classes for JMS bajo el Gestor de seguridad de Java.

Las interfaces `MQSendExit`, `MQReceiveExit` y `MQSecurityExit` que se suministran con las versiones de IBM WebSphere MQ anteriores a la Version 7.0 todavía están 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 405. Los programas de salida de canal escritos en C o C++ se deben almacenar en el directorio mostrado en la Tabla 130 en la página 932.

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 WebSphere MQ classes for JMS” en la página 934.

Especificación de los datos de usuario que se deben pasar a las salidas de canal cuando se utilizan clases WebSphere MQ para 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 JMS de WebSphere MQ o WebSphere 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 JMS de WebSphere MQ o WebSphere 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 WebSphere MQ classes for JMS” en la página 934.

Utilización de una tabla de definiciones de canal de cliente con IBM WebSphere MQ classes for JMS

Una aplicación IBM WebSphere MQ classes for JMS puede utilizar definiciones de canal de conexión de cliente almacenadas en una tabla de definiciones 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 IBM WebSphere MQ classes for JMS puede utilizar definiciones de canal de conexión de cliente que se almacenan en una tabla de definiciones de canal de cliente. Estas definiciones las crean los mandatos de IBM WebSphere MQ Script (MQSC) o IBM WebSphere MQ Programmable Command Format (PCF). Cuando la aplicación crea un objeto Connection, IBM WebSphere MQ classes for JMS busca en la tabla de definiciones de canal de cliente una definición de canal de conexión de cliente adecuada y utiliza la definición de canal 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. 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 JMS de IBM WebSphere MQ , 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.
- Un asterisco (*)

- 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 utiliza la interfaz de cola 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 JMS WebSphere MQ o IBM WebSphere MQ Explorer. Como alternativa, una aplicación puede establecer la propiedad llamando al método setQueueManager() del objeto ConnectionFactory.

Si una aplicación crea un objeto Connection a partir del objeto ConnectionFactory , IBM WebSphere MQ classes for JMS accede a la tabla de definiciones 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, a continuación, utiliza la definición de canal para iniciar un canal MQI en 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 WebSphere MQ classes for JMS encuentra una definición de canal de conexión de cliente adecuada en la tabla de definiciones 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 concreto, tenga en cuenta los puntos siguientes si utiliza SSL (Secure Sockets Layer):

- Un canal MQI utiliza SSL sólo si la definición de canal extraída de la tabla de definiciones de canal de cliente especifica el nombre de una CipherSpec soportada por IBM WebSphere 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 WebSphere MQ classes for JMS sólo utiliza esta información para acceder a los servidores LDAP que contienen CRL.
- Una tabla de definiciones de canal de cliente también puede contener la ubicación de un programa de respuesta OCSP (Online Certificate Status Protocol). IBM WebSphere MQ classes for JMS no puede utilizar la información de OCSP en un archivo de tabla de definición de canal de cliente. Sin embargo, puede configurar OCSP como se describe en la sección Utilización del protocolo de certificados en línea.

Para obtener más información sobre cómo utilizar SSL con una tabla de definiciones de canal de cliente, consulte Utilización del cliente transaccional extendido con canales SSL.

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 definiciones de canal de cliente puede especificar salidas de canal 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 salidas de canal en Java para clases de WebSphere MQ para JMS” en la página 930.

  También se da soporte al uso de salidas de canal escritas en un lenguaje que no sea 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 el cliente JMS para que se vuelva a conectar automáticamente después de una anomalía de red, gestor de colas o servidor.

Utilice las propiedades `CONNECTIONNAMELIST` y `CLIENTRECONNECTOPTIONS` de la clase `MQConnectionFactory` para configurar una conexión de cliente para que se vuelva a conectar automáticamente después de un error de conexión o una solicitud administrativa para volver a conectar las aplicaciones cliente después de detener un gestor de colas.

La lista completa de nombres de conexión en una lista connectionNamesólo es accesible para los métodos set/getconnectionNameList que pueden manejar una lista de nombres de conexión. Los métodos como, por ejemplo, get/setHostname que no manejan listas de nombres, acceden al primer nombre de la lista.

Las conexiones de cliente reconectables automáticamente sólo se vuelven reconectables una vez que se ha establecido la conexión.

El hecho de que una aplicación siga funcionando correctamente después de reconectarse automáticamente depende de su diseño. Lea los temas relacionados para saber cómo diseñar clientes reconectables. Algunos clientes existentes podrían funcionar correctamente sin modificación después de la reconexión automática.

La reconexión automática del cliente no está soportada en WebSphere MQ classes for Java.

Para evitar que todos los clientes conectados a un gestor de colas anómalo se reconecten simultáneamente, los intentos de reconexión se retrasan en intervalos que son parcialmente fijos y parcialmente aleatorios.

De forma predeterminada, la reconexión se lleva a cabo con los intervalos siguientes:

1. El primer intento se realiza después de un retardo inicial de un segundo, más un elemento aleatorio de hasta 250 milisegundos.
2. El segundo intento se realiza dos segundos, más un intervalo aleatorio de hasta 500 milisegundos, después de que el primer intento falle.
3. El tercer intento se realiza cuatro segundos, más un intervalo aleatorio de hasta un segundo, después de que el segundo intento falle.
4. El cuarto intento se realiza ocho segundos, más un intervalo aleatorio de hasta dos segundos, después de que el tercer intento falle.
5. El quinto intento se realiza 16 segundos, más un intervalo aleatorio de hasta cuatro segundos, después de que el cuarto intento falle.
6. El sexto intento, y todos los intentos posteriores se realizan 25 segundos, más un intervalo aleatorio de hasta seis segundos y 250 milisegundos, después de que el intento anterior falle.

Este proceso de reconexión continúa, hasta que el cliente se reconecta correctamente al gestor de colas, o hasta que transcurre el intervalo máximo de reconexión.

Si necesita aumentar los valores predeterminados, para reflejar de forma más precisa la cantidad de tiempo necesaria para que se recupere el gestor de colas, o para que se active el gestor de colas en espera, cambie los valores de retardo en MQCLIENT.INI utilizando el atributo **`ReconDelay`** .

Conceptos relacionados

Reconexión de cliente automatizada

Tareas relacionadas

Configuración de un cliente utilizando un archivo de configuración

Compartición de una conexión TCP/IP en IBM WebSphere 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 dentro del mismo entorno de ejecución Java y que utilizan las clases IBM WebSphere MQ classes for JMS o el adaptador de recursos IBM WebSphere MQ para conectarse a un

gestor de colas utilizando el transporte CLIENT, se pueden realizar para compartir la misma instancia de canal.

Existe una relación de uno a uno entre las instancias de canal y las conexiones TCP/IP. Se crea una conexión TCP/IP para cada 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 sesión JMS creada por una aplicación JMS crea su propia conversación con el gestor de colas.

Cuando se inicia una especificación de activación, el adaptador de recursos IBM WebSphere MQ para JMS 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 enfoque sin garantías para la compartición de conexiones. Por lo tanto, cuando se utiliza un valor SHARECNV mayor que 0 con las clases IBM WebSphere MQ classes for JMS, no se garantiza que una nueva solicitud de conexión siempre comparta una conexión ya establecida.

Cálculo del número de instancias de canal

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 = (<**maxPoolDepth**> + 1)/<**SHARECNV**>

Donde <**maxPoolDepth**> es el valor de la propiedad **maxPoolDepth** y <**SHARECNV**> es el valor de la propiedad **SHARECNV** en el canal que utiliza la especificación de activación.

Otras aplicaciones JMS

Número de instancias de canal = (< conexiones JMS > + < sesiones JMS >)/<**SHARECNV**>

Donde < conexiones JMS > es el número de conexiones creadas por la aplicación, donde < sesiones JMS > es el número de sesiones JMS creadas por la aplicación y <**SHARECNV**> es el valor de la propiedad **SHARECNV** en el canal que utiliza la especificación de activación.

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 WebSphere MQ classes for JMS o el adaptador de recursos de IBM WebSphere MQ classes for JMS.

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; como la propiedad **SHARECNV** en el canal se ha establecido en 0, cada conversación utiliza su propia instancia de canal.

Especificación de un rango de puertos para conexiones de cliente en clases WebSphere MQ para JMS

Utilice la propiedad LOCALADDRESS para especificar un rango de puertos a los que se puede enlazar aplicación.

Cuando una aplicación de WebSphere MQ classes for JMS intenta conectarse a un gestor de colas de WebSphere MQ en modalidad de cliente, un cortafuegos sólo puede permitir las conexiones que se originan en puertos especificados o en un rango de puertos. 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 JMS WebSphere MQ 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 emite una excepción JMSException con una excepción MQException incorporada que contiene el código de razón WebSphere 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).

Puesto que las clases de WebSphere MQ para JMS pueden crear conexiones que no sean las necesarias para una aplicación, considere siempre la posibilidad de especificar un rango de puertos. En general, cada sesión creada por una aplicación requiere un puerto y las clases WebSphere MQ para JMS pueden 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 las clases WebSphere MQ para JMS, puede tener un efecto en 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 clases WebSphere MQ para JMS

Una aplicación WebSphere MQ classes for JMS puede utilizar los recursos de WebSphere MQ para comprimir una cabecera o datos de mensaje.

La compresión de los datos que fluyen en un canal de WebSphere MQ puede mejorar el rendimiento del canal y reducir el tráfico de red. Utilizando la función proporcionada con WebSphere MQ, puede comprimir los datos que fluyen en canales de mensajes y canales 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 WebSphere MQ para JMS especifica las técnicas que se pueden utilizar para comprimir 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.

Colocación asíncrona de mensajes en clases IBM WebSphere MQ para 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 JMS PUTSYNCALLOWED (nombre abreviado-PAALD).

  PUTASYNCALLOWED controla si las aplicaciones JMS pueden transferir mensajes de forma asíncrona, si la cola o tema subyacente que representa el destino JMS, permite esta opción.

- La propiedad de cola o tema IBM WebSphere MQ DEFPRESP (tipo de respuesta de colocación predeterminado).

  DEFPRESP especifica si las aplicaciones que transfieren mensajes a la cola, o publican mensajes en el tema, pueden transferir mensajes de forma asíncrona.

La tabla siguiente muestra los valores posibles para las propiedades PUTASYNCALLOWED y DEFPRESP, y qué valores son necesarios para que esté habilitada la capacidad de transferir mensajes de forma asíncrona:

Tabla 131. Propiedades PUTASYNCALLOWED y DEFPRESP para transferir mensajes de forma asíncrona.

Propiedad de cola WebSphere MQ	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST o AS_Q_DEF o AS_T_DEF**
DEFPRESP=SYNC	Operación de transferencia asíncrona no habilitada	Funcionalidad de puesta en forma asíncrona habilitada	Operación de transferencia asíncrona no habilitada
DEFPRESP=ASYNC	Operación de transferencia asíncrona no habilitada	Funcionalidad de puesta en forma asíncrona habilitada	Funcionalidad de puesta en forma asíncrona habilitada

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 que no es transaccional, la propiedad SENDCHECKCOUNT del objeto *ConnectionFactory* especifica cuántos mensajes se van a enviar, antes de que las clases IBM WebSphere MQ para JMS comprueben que el gestor de colas ha recibido los mensajes de forma segura.

Si una comprobación descubre que uno o más mensajes no se han recibido de forma segura y la aplicación ha registrado un escucha de excepciones con la conexión, IBM WebSphere MQ classes for JMS llama al método `onException()` del escucha de excepciones para pasar una excepción JMS a la aplicación.

La excepción JMS tiene un código de error de JMSWMQ0028 y este código muestra el mensaje siguiente:

```
At least one asynchronous put message failed or gave a warning.
```

La excepción 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.

Utilización de la lectura anticipada con WebSphere MQ classes for JMS

La funcionalidad de lectura anticipada que proporciona WebSphere MQ permite que los mensajes no persistentes que se reciben fuera de una transacción se envíen al IBM WebSphere MQ classes for JMS antes de que una aplicación los solicite. Las IBM WebSphere 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 WebSphere 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 tema subyacente que representa el destino JMS, permite esta opción.
- La propiedad de tema o cola IBM WebSphere 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 de las propiedades READAHEADALLOWED y DEFREADA, y qué valores son necesarios para poder habilitar la función de lectura anticipada:

Tabla 132. Propiedades READAHEADALLOWED y DEFREADA que determinan si se utiliza la lectura anticipada al recibir o examinar mensajes no persistentes fuera de una transacción

Propiedad de destino WebSphere MQ	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST o AS_Q_DEF o AS_T_DEF**
Propiedad de cola WebSphere MQ			
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 WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS devuelven automáticamente el primer mensaje a la aplicación, mediante la llamada a método que haya realizado la aplicación. Las IBM WebSphere 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 WebSphere MQ classes for JMS devolverán el siguiente mensaje del almacenamiento intermedio interno.

Las IBM WebSphere MQ classes for JMS solicitan más mensajes no persistentes del gestor de colas cuando el almacenamiento intermedio interno está vacío.

El almacenamiento intermedio interno que utiliza IBM WebSphere MQ classes for JMS se suprime cuando una aplicación cierra un `MessageConsumero` la sesión JMS con la que está asociado un `MessageListener` .

Para `MessageConsumers`, los mensajes sin procesar del almacenamiento intermedio interno se pierden.

Cuando se utiliza `MessageListeners`, lo que sucede con los mensajes en el almacenamiento intermedio interno depende de la propiedad de destino JMS READAHEADCLOSEPOLICY (nombre abreviado-RACP). El valor predeterminado de la propiedad es DELIVER_ALL, lo 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 se descarten todos los mensajes restantes del almacenamiento intermedio interno.

Publicaciones retenidas en WebSphere MQ classes for JMS

Un cliente WebSphere MQ para JMS puede configurarse para utilizar 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 WebSphere 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 XA en clases WebSphere MQ para JMS

JMS da soporte a transacciones compatibles con XA en enlaces y modalidades de cliente con un gestor de transacciones soportado.

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.

Utilización de una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker

Una aplicación WebSphere MQ para JMS puede utilizar una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker para mensajería de publicación/suscripción. Tanto el intermediario como las clases WebSphere MQ para JMS deben estar configuradas para habilitar una conexión en tiempo real.

Cuando una aplicación utiliza una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker, la aplicación y el intermediario intercambian mensajes utilizando WebSphere MQ Real-Time Transport. En función de la configuración, los mensajes también se pueden entregar a la aplicación utilizando WebSphere MQ Multicast Transport.

Para obtener información sobre cómo una aplicación puede conectarse a un gestor de colas WebSphere MQ y utilizar WebSphere MQ Enterprise Transport para intercambiar mensajes con un intermediario de WebSphere Event Broker o WebSphere Message Broker, consulte la documentación de los releases anteriores de WebSphere MQ classes for JMS. Tenga en cuenta que, para utilizar WebSphere MQ Enterprise Transport, una aplicación debe conectarse a un gestor de colas utilizando una fábrica de conexiones que se ejecute en la modalidad de migración de proveedor de mensajería de WebSphere MQ .

Configuración de un intermediario de WebSphere Event Broker o WebSphere Message Broker para una conexión en tiempo real

Para que una aplicación WebSphere MQ classes for JMS utilice una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker, debe configurar el intermediario creando y desplegando un flujo de mensajes para leer mensajes del puerto TCP/IP en el que el intermediario está a la escucha y publicar los mensajes. En función de sus requisitos, es posible que tenga que configurar el intermediario de formas adicionales.

Para configurar el intermediario, debe crear y desplegar uno de los siguientes flujos de mensajes:

- Un flujo de mensajes que contiene un nodo de proceso de mensajes de flujo Real-timeOptimized
- Un flujo de mensajes que contiene 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 que escuche en el puerto TCP/IP utilizado para conexiones en tiempo real. De forma predeterminada, el número de puerto para conexiones en tiempo real es 1506.

También debe configurar el intermediario si tiene alguno de los requisitos siguientes:

- Si desea que la aplicación se conecte al intermediario utilizando la autenticación SSL (Secure Sockets Layer)
- Si desea que la aplicación se conecte al intermediario utilizando el túnel HTTP
- Si desea que los mensajes se entreguen a un consumidor de mensajes utilizando multidifusión

Para obtener información sobre cómo configurar un intermediario, consulte la *documentación del producto WebSphere Event Broker* o la *documentación del producto WebSphere Message Broker*.

Configuración de WebSphere MQ classes for JMS para una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker

Para que una aplicación WebSphere MQ para JMS utilice una conexión en tiempo real con un intermediario de WebSphere Event Broker o WebSphere Message Broker, las clases WebSphere MQ para JMS deben configurarse estableciendo determinadas propiedades de la fábrica de conexiones. En función de sus requisitos, es posible que sea necesario configurar las clases WebSphere MQ para JMS de formas adicionales.

Para configurar las clases de WebSphere MQ para JMS, se deben establecer las propiedades siguientes de la fábrica de conexiones:

- La propiedad TRANSPORT debe establecerse en DIRECT.

  Sin embargo, para que una aplicación se conecte utilizando el túnel HTTP, la propiedad TRANSPORT debe establecerse en DIRECTHTTP en su lugar. Consulte “Utilización del túnel HTTP” en la página 944.
- La propiedad HOSTNAME debe establecerse en el nombre de host o la dirección IP del sistema en el que se ejecuta el intermediario.
- La propiedad PORT debe establecerse en el número del puerto en el que el intermediario está a la escucha de conexiones en tiempo real.

Una aplicación puede establecer estas propiedades dinámicamente en tiempo de ejecución utilizando las extensiones JMS de IBM o las extensiones JMS de WebSphere MQ . De forma alternativa, si la fábrica de conexiones es un objeto administrado, un administrador puede establecer estas propiedades utilizando la herramienta de administración JMS de WebSphere MQ o WebSphere MQ Explorer.

Para obtener información sobre las propiedades y los métodos utilizados por las aplicaciones para establecer sus valores, consulte Propiedades de objetos IBM WebSphere MQ classes for JMS. Para obtener información sobre cómo utilizar la herramienta de administración JMS de WebSphere MQ , consulte “Utilización de la herramienta de administración JMS de WebSphere MQ” en la página 953. Para obtener información sobre cómo utilizar WebSphere MQ Explorer, consulte la ayuda proporcionada con WebSphere MQ Explorer.

Si tiene alguno de los requisitos siguientes, WebSphere MQ classes for JMS requiere configuración adicional:

- Si desea que una aplicación se conecte al intermediario utilizando la autenticación SSL (Secure Sockets Layer)
- Si desea que una aplicación se conecte al intermediario utilizando el túnel HTTP
- Si desea que una aplicación se conecte al intermediario a través de un servidor proxy
- Si desea que los mensajes se entreguen a un consumidor de mensajes utilizando multidifusión

En las secciones siguientes se describe cómo configurar las clases de WebSphere MQ para JMS para cada uno de estos requisitos.

Utilización de la autenticación SSL (Secure Sockets Layer)

La autenticación SSL se puede utilizar en una conexión en tiempo real con un intermediario. Sólo se da soporte a la autenticación para este tipo de conexión. No puede utilizar SSL para cifrar y descifrar los datos de mensaje que fluyen entre la aplicación y el intermediario o para detectar la manipulación indebida de los datos.

Tenga en cuenta la diferencia entre esta situación y la que existe cuando una aplicación se conecta a un gestor de colas en modalidad de cliente. En este último caso, puede utilizar el soporte SSL de WebSphere MQ para cifrar y descifrar los datos de mensaje que fluyen entre la aplicación y el gestor de colas y para detectar la manipulación indebida de los datos, así como para proporcionar autenticación.

Si desea proteger los datos de mensaje en una conexión en tiempo real con un intermediario, puede utilizar la función proporcionada por el intermediario en su lugar. Puede asignar un valor de calidad de protección (QoP) a cada tema con mensajes que desee proteger. Por lo tanto, puede seleccionar un nivel diferente de protección de mensajes para cada tema. Para obtener más información sobre la protección de mensajes proporcionada por un intermediario, consulte la *documentación del producto WebSphere Event Broker* o la *documentación del producto WebSphere Message Broker*.

Para utilizar la autenticación SSL en una conexión en tiempo real con un intermediario, la propiedad DIRECTAUTH de la fábrica de conexiones debe establecerse en CERTIFICATE.

Si desea utilizar SSL para la autenticación mutua, la propiedad Tipo de protocolo de autenticación del intermediario debe especificar la opción R para SSL simétrico. Si desea utilizar SSL sólo para autenticar el intermediario, la propiedad Tipo de protocolo de autenticación del intermediario debe especificar la opción S para SSL asimétrico. Pero, en este caso, la aplicación debe conectarse al intermediario llamando a createConnection() con un ID de usuario y una contraseña como parámetros, como en el ejemplo siguiente:

```
factory.createConnection("user1", "user1pw");
```

A continuación, el intermediario utiliza el ID de usuario y la contraseña, en lugar de SSL, para autenticar la aplicación. Para obtener más información sobre cómo configurar el intermediario para la autenticación SSL, consulte la *documentación del producto WebSphere Event Broker* o la *documentación del producto WebSphere Message Broker*.

Notas:

1. El valor de la propiedad DIRECTAUTH determina si se utiliza la autenticación SSL en una conexión en tiempo real con un intermediario, no el valor de la propiedad SSLCIPHERSUITE.
2. Cuando se utiliza la autenticación SSL en una conexión en tiempo real con un intermediario, las propiedades SSLPEERNAME y SSLCRL se utilizan para realizar las mismas comprobaciones que las realizadas cuando una aplicación se conecta a un gestor de colas en modalidad de cliente.
3. WebSphere MQ classes for JMS puede utilizar la misma configuración de almacén de claves y almacén de confianza de JSSE (Java Secure Socket Extension) para proporcionar el soporte SSL en cualquiera de las situaciones siguientes:
   - Cuando una aplicación utiliza una conexión en tiempo real con un intermediario
   - Cuando una aplicación se conecta a un gestor de colas en modalidad de cliente

Utilización del túnel HTTP

Una aplicación WebSphere MQ para JMS puede conectarse a un intermediario utilizando el túnel HTTP, lo que significa que la aplicación se conecta al intermediario utilizando el protocolo HTTP como si se conectara a un sitio web.

Para utilizar el túnel HTTP en una conexión en tiempo real con un intermediario, la propiedad TRANSPORT de la fábrica de conexiones debe establecerse en DIRECTHTTP.

El túnel HTTP no se puede utilizar junto con la autenticación SSL, conectándose a través de un servidor proxy o entregando mensajes utilizando multidifusión. La versión soportada del protocolo HTTP es 1.0. HTTP versión 1.1 no está soportado.

Conexión a través de un servidor proxy

Una aplicación WebSphere MQ para JMS puede utilizar una conexión en tiempo real con un intermediario conectándose a través de un servidor proxy. WebSphere MQ classes for JMS se conecta directamente al servidor proxy y utiliza el protocolo Internet definido en RFC 2817 para solicitar al servidor proxy que reenvíe la solicitud de conexión al intermediario.

Para conectarse a un intermediario a través de un servidor proxy, se deben establecer las propiedades siguientes de la fábrica de conexiones:

- La propiedad PROXYHOSTNAME debe establecerse en el nombre de host o la dirección IP del sistema en el que se ejecuta el servidor proxy.
- La propiedad PROXYPORT debe establecerse en el número del puerto en el que el servidor proxy está a la escucha.

Si la propiedad PROXYHOSTNAME no está establecida, o está establecida en la serie vacía, WebSphere MQ classes for JMS intenta conectarse directamente al intermediario utilizando sólo las propiedades HOSTNAME y PORT, y no intenta conectarse a través de un servidor proxy.

Entrega de mensajes utilizando multidifusión

Utilizando una conexión en tiempo real con un intermediario, los mensajes se pueden entregar a un consumidor de mensajes utilizando multidifusión.

Para habilitar la multidifusión, la propiedad MULTICAST del objeto Topic debe establecerse en la opción de multidifusión necesaria. De forma alternativa, si la propiedad MULTICAST del objeto Topic se establece en ASCF, la propiedad MULTICAST de la fábrica de conexiones se debe establecer en la opción de multidifusión necesaria.

WebSphere MQ classes for JMS da soporte a los protocolos de multidifusión PTL (Packet Transfer Layer) y PGM (Pragmático General Multicast), e incluye soporte para ambas implementaciones del protocolo PGM, PGM/IP y PGM UDP encapsulados. Sin embargo, el soporte de PGM/IP sólo está disponible en las plataformas siguientes:

- AIX (sólo 32 bits)
- Linux (plataforma x86)
- Linux (plataforma zSeries , sólo de 32 bits)
- Solaris SPARC (sólo 32 bits)
- Windows (sólo 32 bits)
- z/OS

WebSphere MQ classes for JMS Application Server Facilities

En este tema se describe cómo WebSphere MQ classes for JMS implementa la clase ConnectionConsumer y la funcionalidad avanzada en la clase Session. También describe la función de una agrupación de sesiones de servidor.

WebSphere MQ classes for JMS da soporte a ASF (Application Server Facilities) que se especifican en la *especificación Java Message Service, Versión 1.1* (consulte el sitio web Java de Sun en https://java.sun.com). Esta especificación identifica tres funciones dentro de este modelo de programación:

- **El proveedor JMS** proporciona ConnectionConsumer y funcionalidad de sesión avanzada.
- **El servidor de aplicaciones**, que proporciona las funciones ServerSessionPool y ServerSession.
- **La aplicación cliente** utiliza la funcionalidad que proporcionan el proveedor 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.

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 JMS permite a un servidor de aplicaciones integrarse estrechamente con una implementación 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 JMS hace que los mensajes estén disponibles para estas hebras. Un servidor de aplicaciones con reconocimiento JMS (como WebSphere Application Server) puede utilizar esta característica para proporcionar funcionalidad de mensajería de alto nivel, como por ejemplo beans controlados por mensajes.

Las aplicaciones normales no utilizan ConnectionConsumer, pero los clientes JMS expertos pueden utilizarlo. Para este tipo de clientes, ConnectionConsumer proporciona un método de alto rendimiento para entregar mensajes simultáneamente a una agrupación de hebras. Cuando llega un mensaje a una cola o un tema, JMS selecciona una hebra de la agrupación y le entrega un lote de mensajes. Para ello, JMS ejecuta un método MessageListenerasociado `onMessage()` .

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 946
- “Principios generales de la mensajería de publicación/suscripción utilizando ASF” en la página 947
- “Eliminación de mensajes de la cola en ASF” en la página 948
- Manejo de mensajes no entregables en ASF. Consulte “Manejo de mensajes no entregables en IBM WebSphere MQ classes for JMS” en la página 908.

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 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 WebSphere 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 WebSphere 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 948). Para establecer este comportamiento en una de estas dos maneras:

- Utilice la herramienta de administración JMS para establecer la fábrica QueueConnectionen 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 WebSphere MQ , tenga en cuenta los puntos siguientes:

- 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 obtener detalles, consulte la documentación de WebSphere MQ para su plataforma específica.
- 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 la publicación Referencia de 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 WebSphere MQ classes for JMS está en modalidad de migración de proveedor de mensajería de WebSphere MQ , se utiliza una cola independiente para suscripciones 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 WebSphere MQ classes for JMS está en modalidad de migración de proveedor de mensajería de WebSphere MQ , la propiedad CCDSUB del tema especifica la cola que se utilizará para las 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 WebSphere MQ , tenga en cuenta los puntos siguientes:

- 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 más detalles, consulte la documentación de WebSphere MQ para su plataforma.
- 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, es posible que JMS tenga que eliminar mensajes de la cola en una serie de situaciones.

Estas situaciones son las siguientes:

Mensaje con formato incorrecto
Puede llegar un mensaje que JMS no puede analizar.

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. Éste 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.

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 vuelve a poner en cola en la cola de mensajes no entregados, va precedido de una cabecera de mensajes no entregados (MQDLH) de WebSphere MQ . 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)`
- PutApplEl nombre 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 950. Encontrará la descripción de otros campos en MQDLH - Cabecera de mensaje no entregado.

Manejo de mensajes dañados en ASF

Dentro de los recursos de servidor de aplicaciones, el manejo de mensajes con formato incorrecto se maneja de forma ligeramente diferente al resto de las clases de WebSphere MQ para JMS.

Para obtener información sobre el manejo de mensajes con formato incorrecto en las clases WebSphere MQ para JMS, consulte “Manejo de mensajes no entregables en IBM WebSphere MQ classes for JMS” en la página 908.

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.

El proveedor JMS incorporado en WebSphere Application Server, Versión 5.0 y Versión 5.1 maneja los mensajes con formato incorrecto de una forma diferente a la que se acaba de describir para WebSphere MQ classes for JMS. Para obtener información sobre cómo maneja el proveedor JMS incorporado los mensajes con formato incorrecto, consulte la documentación del producto WebSphere Application Server.

Tratamiento de errores

Esta sección describe diversos aspectos del manejo de errores, incluidos "Recuperación para condiciones de error en los ASF" en la página 950 y "Códigos de razón y de información de retorno en ASF" en la página 950.

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)

Motivo

El mensaje ha alcanzado al umbral de restitución definido para la QLOCAL, pero no se ha definido ninguna cola de retirada.

En plataformas en las que no puede definir la cola de restitución, el mensaje ha alcanzado el umbral de restitución definido por JMS de 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)

Motivo
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)

Motivo
JMS no puede interpretar el mensaje en la cola.

Acción
Determine el origen del mensaje. JMS normalmente entrega 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 Códigos de razón de 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 165 en la página 952 resume los principios de las funciones de ServerSessionPool y ServerSession.

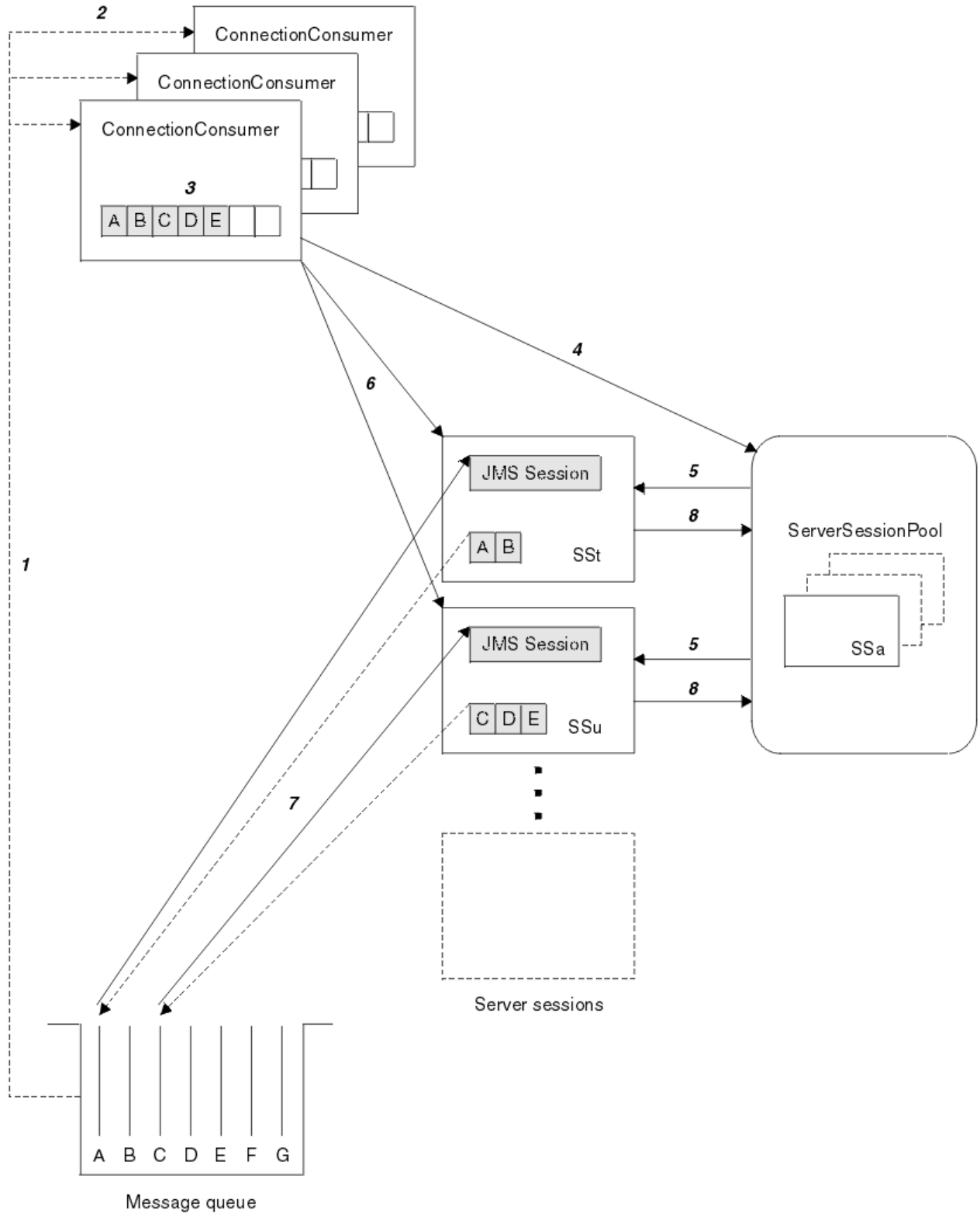


Figura 165. Funciones de ServerSessionPool y ServerSession

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. Los pasa al método onMessage desde el MessageListener que está asociado con la sesión 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.

Utilización de la herramienta de administración JMS de WebSphere MQ

Utilice la herramienta de administración para definir las propiedades de ocho tipos de WebSphere MQ clases para objeto JMS y para almacenarlas en un espacio de nombres JNDI. Las aplicaciones pueden luego utilizar JNDI para recuperar estos objetos administrados del espacio de nombres.

Las clases JMS de WebSphere MQ que puede administrar utilizando la herramienta son:

- MQConnectionFactory
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

Para obtener detalles de estos objetos, consulte “Administración de objetos JMS” en la página 958 .

Los tipos y valores de propiedad que necesita para utilizar esta herramienta se listan en Propiedades de objetos de IBM WebSphere MQ classes for JMS.

La herramienta también permite a los administradores manipular subcontextos de espacio de nombres de directorio dentro de JNDI. Consulte “Manipulación de subcontextos con la herramienta de administración JMS de WebSphere MQ” en la página 957.

También puede crear y configurar objetos administrados JMS con WebSphere MQ Explorer.

Invocación de la herramienta de administración de IBM WebSphere MQ classes for JMS

La herramienta de administración tiene una interfaz de línea de mandatos. Puede utilizarlo de forma interactiva o utilizarlo para iniciar un proceso por lotes.

La modalidad interactiva proporciona un indicador de mandatos en el que puede entrar mandatos de administración. En la modalidad de proceso por lotes, el mandato para iniciar la herramienta incluye el nombre de un archivo que contiene un script de mandatos de administración.

Modalidad interactiva

Para iniciar la herramienta en modalidad interactiva, entre el mandato:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

donde:

-t
Habilita el rastreo (el valor predeterminado es que el rastreo esté desactivado)

El archivo de rastreo se genera en "%MQ_JAVA_DATA_PATH%"\errors (Windows) o /var/mqm/trace (UNIX). El nombre del archivo de rastreo tiene el formato:

mqjms_*PID*.trc

donde *PID* es el ID de proceso de la JVM.

-v
Produce salida detallada (el valor predeterminado es salida concisa)

-cfg nombre_archivo_config
Indica un archivo de configuración alternativo. Si se omite este parámetro, se utiliza el archivo de configuración predeterminado, JMSAdmin.config. (Consulte "Configuración de la herramienta de administración JMS" en la página 954)

Se visualiza un indicador de mandatos, lo que indica que la herramienta está preparada para aceptar mandatos de administración. Este indicador aparece inicialmente como:

```
InitCtx>
```

indicando que el contexto actual (es decir, el contexto JNDI al que hacen referencia actualmente todas las operaciones de denominación y directorio) es el contexto inicial definido en el parámetro de configuración PROVIDER_URL (consulte "Configuración de la herramienta de administración JMS" en la página 954).

A medida que se atraviesa el espacio de nombres de directorio, el indicador cambia para reflejarlo, por lo que el indicador siempre muestra el contexto actual.

Modalidad de proceso por lotes

Para iniciar la herramienta en modalidad de proceso por lotes, especifique el mandato:

```
JMSAdmin <test.scp
```

donde *test.scp* es un archivo de script que contiene mandatos de administración (consulte "Mandatos de administración en la herramienta de administración JMS de WebSphere MQ" en la página 956). El último mandato del archivo debe ser el mandato END.

Configuración de la herramienta de administración JMS

La herramienta de administración JMS de WebSphere MQ utiliza un archivo de configuración para establecer los valores de determinadas propiedades. Se proporciona un archivo de ejemplo, que puede adaptar a su sistema.

El archivo de configuración es un archivo de texto sin formato que consta de un conjunto de pares de clave-valor, separados por el signo igual (=). Esto se muestra en el ejemplo siguiente:

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(Un # en la primera columna de la línea indica un comentario, o una línea que no se utiliza.)

Se proporciona un archivo de configuración de ejemplo con WebSphere MQ. El archivo se denomina JMSAdmin.configy se encuentra en el directorio <MQ_JAVA_INSTALL_PATH>/bin . Edite este archivo para que se ajuste a la configuración del sistema.

Configure la herramienta de administración con valores para las propiedades siguientes:

INITIAL_CONTEXT_FACTORY
El proveedor de servicios que utiliza la herramienta. Los valores soportados para esta propiedad son los siguientes:

- com.sun.jndi.ldap.LdapCtxFactory (para LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (para el contexto de sistema de archivos)

También puede utilizar una fábrica InitialContextque no esté en la lista anterior. Consulte "Utilización de una fábrica InitialContextno listada con la herramienta de administración JMS de WebSphere MQ" en la página 955 para obtener más detalles.

PROVIDER_URL
El URL del contexto inicial de la sesión; la raíz de todas las operaciones JNDI realizadas por la herramienta. Se admiten dos formatos de esta propiedad:

- ldap://nombre_host/nombre_contexto
- file:[unidad:]/nombre_vía_acceso

El formato del URL de LDAP puede variar en función del proveedor de LDAP. Consulte la documentación de LDAP para obtener más información.

AUTENTICACIÓN_SEGURIDAD
Indica si JNDI pasa credenciales de seguridad al proveedor de servicios. Esta propiedad sólo se utiliza cuando se utiliza un proveedor de servicios LDAP. Esta propiedad puede tomar uno de estos tres valores:

- none (autenticación anónima)
- simple (autenticación simple)
- CRAM-MD5 (mecanismo de autenticaciónCRAM-MD5)

Si no se proporciona un valor válido, la propiedad toma el valor predeterminado none. Consulte "Configuración de la seguridad para la herramienta de administración JMS" en la página 956 para obtener más detalles sobre la seguridad con la herramienta de administración.

Estas propiedades se establecen en un archivo de configuración. Al invocar la herramienta, puede especificar esta configuración utilizando el parámetro de línea de mandatos `-cfg` , tal como se describe en "Invocación de la herramienta de administración de IBM WebSphere MQ classes for JMS" en la página 953. Si no especifica un nombre de archivo de configuración, la herramienta intenta cargar el archivo de configuración predeterminado (`JMSAdmin.config`). Busca este archivo primero en el directorio actual y, a continuación, en el directorio `<MQ_JAVA_INSTALL_PATH>/bin` , donde `<MQ_JAVA_INSTALL_PATH>` es la vía de acceso a las clases de WebSphere MQ para la instalación de JMS.

Utilización de una fábrica InitialContextno listada con la herramienta de administración JMS de WebSphere MQ

Se da soporte a dos valores de fábrica InitialContext. Puede utilizar otros contextos JNDI estableciendo parámetros en el archivo de configuración de administración JMS.

Puede utilizar la herramienta de administración para conectarse a contextos JNDI distintos de los listados en "Configuración de la herramienta de administración JMS" en la página 954 utilizando tres parámetros definidos en el archivo de configuración JMSAdmin.

Para utilizar una InitialContextFactory diferente:

1. Establezca la propiedad INITIAL_CONTEXT_FACTORY en el nombre de clase necesario.
2. Defina el comportamiento de la fábrica InitialContextutilizando las propiedades USE_INITIAL_DIR_CONTEXT, NAME_PREFIX y NAME_READABILITY_MARKER.

Los valores de estas propiedades se describen en los comentarios del archivo de configuración de ejemplo.

No es necesario definir las tres propiedades listadas aquí si utiliza uno de los valores INITIAL_CONTEXT_FACTORY soportados. Sin embargo, puede darles valores para alterar temporalmente los valores predeterminados del sistema. Si omite una o más de las tres propiedades de

InitialContextFactory, la herramienta de administración proporciona valores predeterminados adecuados basándose en los valores de las otras propiedades.

Configuración de la seguridad para la herramienta de administración JMS

Utilice la propiedad SECURITY_AUTHENTICATION para determinar si las credenciales de seguridad se pasan al proveedor de servicios.

La propiedad SECURITY_AUTHENTICATION se describe en "Configuración de la herramienta de administración JMS" en la página 954. Su efecto es el siguiente:

- Si establece este parámetro en none, JNDI no pasa ninguna credencial de seguridad al proveedor de servicios y se realiza una *autenticación anónima*.
- Si establece el parámetro en `simple` o `CRAM-MD5`, las credenciales de seguridad se pasan a través de JNDI al proveedor de servicios subyacente. Estas credenciales de seguridad tienen el formato de un nombre distinguido de usuario (DN de usuario) y una contraseña.

Si se requieren credenciales de seguridad, se le solicitarán cuando se inicialice la herramienta. Evite esto estableciendo las propiedades PROVIDER_USERDN y PROVIDER_PASSWORD en el archivo de configuración JMSAdmin.

Nota: Si no utiliza estas propiedades, el texto escrito, *incluida la contraseña*, se refleja en la pantalla. Esto puede tener implicaciones de seguridad.

La herramienta no realiza ninguna autenticación por sí misma; la tarea se delega en el servidor LDAP. El administrador del servidor LDAP debe configurar y mantener los privilegios de acceso a las distintas partes del directorio. Consulte la documentación de LDAP para obtener más información. Si la autenticación falla, la herramienta muestra un mensaje de error apropiado y finaliza.

Encontrará información más detallada sobre la seguridad y JNDI en la documentación del sitio web Java de Sun (https://java.sun.com).

Mandatos de administración en la herramienta de administración JMS de WebSphere MQ

La herramienta de administración acepta mandatos que constan de un verbo de administración y sus parámetros correspondientes.

Cuando se visualiza el indicador de mandatos, la herramienta está lista para aceptar mandatos. Los mandatos de administración tienen generalmente el formato siguiente:

```
verb [param]*
```

donde **verb** es uno de los verbos de administración listados en Tabla 133 en la página 956. Todos los mandatos válidos contienen un verbo, que aparece al principio del mandato en su forma estándar o abreviada.

Los parámetros que un verbo puede tomar dependen del verbo. Por ejemplo, el verbo END no puede tomar ningún parámetro, pero el verbo DEFINE puede tomar cualquier número de parámetros. Los detalles de los verbos que toman al menos un parámetro se describen en los temas relacionados.

Tabla 133. Verbos de administración

Verbo	Formato abreviado	Descripción
ALTER	ALT	Cambiar al menos una de las propiedades de un objeto administrado
DEFINE	DEF	Crear y almacenar un objeto administrado, o crear un subcontexto
DISPLAY	DIS	Mostrar las propiedades de uno o más objetos administrados almacenados, o el contenido del contexto actual

Tabla 133. Verbos de administración (continuación)		
Verbo	**Formato abreviado**	**Descripción**
Suprimir	SUP	Eliminar uno o más objetos administrados del espacio de nombres, o eliminar un subcontexto vacío
CHANGE	CHG	Modificar el contexto actual, permitiendo al usuario desplazarse a cualquier lugar del espacio de nombres de directorio bajo el contexto inicial (pendiente del permiso de seguridad)
COPY	CP	Hacer una copia de un objeto administrado almacenado y almacenarlo con un nombre alternativo
MOVE	MV	Modificar el nombre con el que se ha almacenado un objeto administrado
END		Cerrar la herramienta de administración

Los nombres de los verbos no son sensibles a las mayúsculas y minúsculas.

Normalmente, para terminar los mandatos, pulse la tecla de retorno de carro. Sin embargo, puede alterar temporalmente esto escribiendo el signo más (+) directamente antes del retorno de carro. Esto le permite entrar mandatos multilínea, tal como se muestra en el siguiente ejemplo:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Las líneas que empiezan por cualquiera de los caracteres siguientes se tratan como comentarios y se ignoran: * #/.

Manipulación de subcontextos con la herramienta de administración JMS de WebSphere MQ

Utilice los verbos **CHANGE**, **DEFINE**, **DISPLAY** y **DELETE** para manipular subcontextos de espacio de nombres de directorio.

El uso de estos verbos se describe en Tabla 134 en la página 957.

Tabla 134. Sintaxis y descripción de los mandatos que se utilizan para manipular subcontextos	
Sintaxis del mandato	**Descripción**
DEFINE CTX(nombreContexto)	Intenta crear un subcontexto hijo del contexto actual, con el nombre nombreContexto. No se ejecuta correctamente si se produce una violación de la seguridad, si el subcontexto ya existe o si el nombre proporcionado no es válido.
DISPLAY CTX	Muestra el contenido del contexto actual. Los objetos administrados se anotan con a, los subcontextos con `[D]`. También se muestra el tipo Java de cada objeto.
DELETE CTX(nombreContexto)	Intenta suprimir el contexto hijo con el nombre nombreContexto del contexto actual. No se ejecuta correctamente si no se encuentra el contexto, si el contexto no está vacío o si se produce una violación de la seguridad.

Tabla 134. Sintaxis y descripción de los mandatos que se utilizan para manipular subcontextos (continuación)	
Sintaxis del mandato	**Descripción**
CHANGE CTX(nombreContexto)	Modifica el contexto actual para que ahora haga referencia al contexto hijo con el nombre nombreContexto. Se puede proporcionar uno de los dos siguientes valores especiales de nombreContexto: **=UP** se traslada al contexto padre del contexto actual **=INIT** se traslada directamente al contexto inicial No se ejecuta correctamente si el contexto especificado no existe o si hay una violación de la seguridad.

Administración de objetos JMS

Esta sección describe los ocho tipos de objeto que la herramienta de administración puede manejar. Incluye detalles sobre cada una de sus propiedades configurables y los verbos que pueden manipularlas.

También puede crear y configurar objetos administrados JMS con WebSphere MQ Explorer.

Tipos de objeto JMS

La tabla muestra los ocho tipos de objetos administrados.

La columna Palabra clave muestra las series que puede sustituir por *TYPE* en los mandatos que se muestran en la Tabla 136 en la página 959.

Tabla 135. Los tipos de objeto JMS que maneja la herramienta de administración		
Tipo de objeto	**Palabra clave**	**Descripción**
MQConnectionFactory	CF	La implementación de WebSphere MQ de la interfaz ConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en los dominios punto a punto y de publicación/suscripción.
MQQueueConnectionFactory	QCF	La implementación de WebSphere MQ de la interfaz de fábrica QueueConnectionde JMS. Representa un objeto de fábrica para crear conexiones en el dominio punto a punto.
MQTopicConnectionFactory	TCF	La implementación de WebSphere MQ de la interfaz de fábrica TopicConnectionde JMS. Representa un objeto de fábrica para crear conexiones en el dominio de publicación/ suscripción.
MQQueue	Q	La implementación de WebSphere MQ de la interfaz de cola JMS. Representa un destino para los mensajes en el dominio punto a punto.

Tabla 135. Los tipos de objeto JMS que maneja la herramienta de administración (continuación)		
Tipo de objeto	**Palabra clave**	**Descripción**
MQTopic	T	La implementación de WebSphere MQ de la interfaz de tema JMS. Representa un destino para los mensajes en el dominio de publicación/suscripción.
MQXAConnectionFactory[“1” en la página 959]	XACF	La implementación de WebSphere MQ de la interfaz JMS XAConnectionFactory . Representa un objeto de fábrica para crear conexiones en los dominios punto a punto y de publicación/suscripción, y donde las conexiones utilizan las versiones XA de las clases JMS.
MQXAQueueConnectionFábrica[“1” en la página 959]	XAQCF	La implementación de WebSphere MQ de la interfaz de fábrica XAQueueConnectionde JMS. Representa un objeto de fábrica para crear conexiones en el dominio punto a punto que utilizan las versiones XA de las clases JMS.
MQXATopicConnectionFábrica[“1” en la página 959]	XATCF	La implementación de WebSphere MQ de la interfaz JMS XATopicConnectionFactory. Representa un objeto de fábrica para crear conexiones en el dominio de publicación/ suscripción que utilizan las versiones XA de las clases JMS.
Nota: 1. Estas clases están destinadas a los proveedores de servidores de aplicaciones. Es poco probable que sean directamente útiles a los programadores de aplicaciones.		

Verbos utilizados con objetos JMS

Puede utilizar los verbos `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY` y `MOVE` para manipular objetos administrados en el espacio de nombres de directorio.

En la Tabla 136 en la página 959 se resume el uso de estos verbos. Sustituya *TYPE* por la palabra clave que representa el objeto administrado necesario, tal como se lista en Tabla 135 en la página 958.

Tabla 136. Sintaxis y descripción de los mandatos que se utilizan para manipular objetos administrados	
Sintaxis del mandato	**Descripción**
ALTER *TYPE*(nombre) [propiedad] *	Intenta actualizar las propiedades del objeto administrado con las suministradas. No se ejecuta correctamente si se produce una violación de la seguridad, si no se puede encontrar el objeto especificado o si las nuevas propiedades suministradas no son válidas.
DEFINE *TYPE*(nombre) [propiedad] *	Intenta crear un objeto administrado de tipo *TYPE* con las propiedades proporcionadas y almacenarlo bajo el nombre `name` en el contexto actual. No se ejecuta correctamente si se produce una violación de la seguridad, si el nombre suministrado no es válido o ya existe un objeto con ese nombre, o si las propiedades suministradas no son válidas.

Tabla 136. Sintaxis y descripción de los mandatos que se utilizan para manipular objetos administrados (continuación)

Sintaxis del mandato	**Descripción**
DISPLAY *TYPE*(nombre)	Muestra las propiedades del objeto administrado de tipo *TYPE*, enlazado bajo el nombre name en el contexto actual. No se ejecuta correctamente si el objeto no existe o si se produce una violación de la seguridad.
DELETE *TYPE*(nombre)	Intenta eliminar el objeto administrado de tipo *TYPE*, que tiene el nombre name, del contexto actual. No se ejecuta correctamente si el objeto no existe o si se produce una violación de la seguridad.
COPY *TYPE*(nameA) *TYPE*(nameB)	Realiza una copia del objeto administrado de tipo *TYPE*, con el nombre nameA, nombrando la copia nameB. Todo esto se produce dentro del ámbito del contexto actual. No se ejecuta correctamente si el objeto que se va a copiar no existe, si existe un objeto con el nombre nombreB o si se produce una violación de la seguridad.
MOVE *TYPE*(nameA) *TYPE*(nameB)	Mueve (renombra) el objeto administrado de tipo *TYPE*, que tiene el nombre nameA, a nameB. Todo esto se produce dentro del ámbito del contexto actual. No se ejecuta correctamente si el objeto que se va a mover no existe, si existe un objeto con el nombre nombreB o si se produce una violación de la seguridad.

Creación de objetos con la herramienta de administración JMS de WebSphere MQ

Cree objetos y almacénelos en un espacio de nombres JNDI utilizando el mandato DEFINE,

Utilice la sintaxis de mandato siguiente:

```
DEFINE TYPE(name) [property]*
```

Es decir, el verbo DEFINE , seguido de una referencia de objeto administrado *TYPE*(name) , seguido de cero o más *propiedades* (consulte Propiedades de objetos IBM WebSphere MQ classes for JMS).

Consideraciones de denominación LDAP para objetos JMS

Para almacenar los objetos en un entorno LDAP, debe asignarles nombres que sigan ciertas convenciones. La herramienta de administración puede ayudarle a cumplir los convenios de denominación añadiendo un prefijo predeterminado.

Un convenio de denominación es que los nombres de objeto y subcontexto deben incluir un prefijo, como cn= (nombre común) o ou= (unidad organizativa).

La herramienta de administración simplifica el uso de los proveedores de servicio LDAP al permitir hacer referencia a nombres de contexto y de objeto sin un prefijo. Si no se proporciona un prefijo, la herramienta añade automáticamente un prefijo predeterminado al nombre especificado. Para LDAP, este prefijo es cn=.

Puede cambiar el prefijo predeterminado estableciendo la propiedad NAME_PREFIX en el archivo de configuración JMSAdmin, tal como se describe en “Utilización de una fábrica InitialContextno listada con la herramienta de administración JMS de WebSphere MQ” en la página 955.

Esto se muestra en el ejemplo siguiente.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX
```

```
Contents of InitCtx

  a  cn=testQueue              com.ibm.mq.jms.MQQueue

 1 Object(s)
   0 Context(s)
   1 Binding(s), 1 Administered
```

Aunque el nombre de objeto proporcionado (`testQueue`) no tiene un prefijo, la herramienta añade automáticamente uno para garantizar la conformidad con el convenio de denominación LDAP. Del mismo modo, el envío del mandato `DISPLAY Q(testQueue)` también hace que se añada este prefijo.

Es posible que tenga que configurar el servidor LDAP para almacenar objetos Java. Para obtener información para ayudarle con esta configuración, consulte la documentación del servidor LDAP.

Condiciones de error de ejemplo que crean un objeto JMS

Pueden surgir una serie de condiciones de error comunes cuando se crea un objeto.

Los siguientes son ejemplos de estas condiciones de error:

CipherSpec correlacionada con CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

Propiedad no válida para el objeto

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

Tipo no válido para valor de propiedad

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

Conflicto de propiedades - cliente/enlaces

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

Conflicto de propiedades - Inicialización de salida

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

Valor de la propiedad fuera del rango válido

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

Propiedad desconocida

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

A continuación se muestran ejemplos de condiciones de error que pueden surgir en Windows al buscar objetos administrados JNDI desde una aplicación JMS.

1. Si utiliza el proveedor JNDI de WebSphere , com.ibm.websphere.naming.WsnInitialContextFactory, debe utilizar una barra inclinada (/) para acceder a los objetos administrados definidos en

subcontextos; por ejemplo, jms/MyQueueName. Si utiliza una barra inclinada invertida (\), se genera una excepción InvalidNameException.

2. Si utiliza el proveedor JNDI de Sun, com.sun.jndi.fscontext.RefFSContextFactory, debe utilizar una barra inclinada invertida (\) para acceder a los objetos administrados definidos en subcontextos; por ejemplo, ctx1\\fred. Si utiliza una barra inclinada (/), se genera una excepción NameNotFoundException.

Utilización de la configuración de WebSphere MQ Explorer for JMS

Utilice la interfaz gráfica de usuario de IBM WebSphere MQ Explorer para crear objetos JMS a partir de objetos WebSphere MQ y objetos WebSphere MQ a partir de objetos JMS, así como para administrar y supervisar otros objetos WebSphere MQ .

Antes de empezar

Antes de crear y configurar objetos administrados JMS con WebSphere MQ Explorer, añada un contexto inicial para definir la raíz del espacio de nombres JNDI en el que se almacenan los objetos JMS en el servicio de nombres y directorios. Para obtener más información, consulte la asistencia al usuario de IBM WebSphere MQ Explorer para objetos administrados JMS.

Acerca de esta tarea

Puede realizar las tareas siguientes con el Explorador de IBM WebSphere MQ , ya sea contextualmente desde un objeto existente en el Explorador de IBM WebSphere MQ , o desde un asistente para crear un objeto nuevo. Consulte la ayuda de WebSphere MQ Explorer para obtener ejemplos de la asistencia al usuario de WebSphere MQ Explorer para algunas tareas típicas.

Procedimiento

- Cree una fábrica de conexiones JMS a partir de cualquiera de los siguientes objetos WebSphere MQ :
  a) Un gestor de colas de WebSphere MQ , ya sea en el sistema local o en un sistema remoto.
  b) Un canal de WebSphere MQ
  c) Un escucha de WebSphere MQ
- Añadir un gestor de colas WebSphere MQ a WebSphere MQ Explorer utilizando una fábrica de conexiones JMS
- Crear una cola JMS a partir de una cola WebSphere MQ
- Crear una cola WebSphere MQ a partir de una cola JMS
- Crear un tema JMS a partir de un tema WebSphere MQ , que puede ser un objeto WebSphere MQ o un tema dinámico
- Crear un tema WebSphere MQ a partir de un tema JMS

Utilización del paquete de cabeceras de WebSphere MQ

El paquete WebSphere MQ Headers proporciona un conjunto de interfaces de ayudante y clases que puede utilizar para manipular las cabeceras WebSphere MQ de un mensaje. Normalmente, se utiliza el paquete WebSphere MQ Headers porque desea realizar servicios administrativos utilizando el servidor de mandatos (utilizando mensajes PCF (Programmable Command Format)).

Acerca de esta tarea

El paquete WebSphere MQ Headers se encuentra en los paquetes `com.ibm.mq.headers` y `com.ibm.mq.pcf` . Puede utilizar este recurso para las dos API alternativas que WebSphere MQ proporciona para su uso en aplicaciones Java:

- Clases WebSphere MQ para Java (también denominadas WebSphere MQ Headers Base Java).

- Clases de WebSphere MQ para Java Message Service (clases de WebSphere MQ para JMS, también denominadas WebSphere MQ JMS).

WebSphere MQ Las aplicaciones Java base suelen manipular objetos MQMessage, y las clases de soporte de cabeceras pueden interactuar directamente con estos objetos, ya que entienden de forma nativa las interfaces Java base de WebSphere MQ .

En WebSphere MQ JMS, la carga útil de un mensaje suele ser una serie o un objeto de matriz de bytes, que se puede manipular con secuencias DataInput y DataOutput . El paquete de cabeceras de WebSphere MQ se puede utilizar para interactuar con estas corrientes de datos y es adecuado para manipular cualquier mensaje de MQ que envíen y reciban las aplicaciones JMS de WebSphere MQ .

Por lo tanto, aunque el paquete de cabeceras WebSphere MQ contiene referencias al paquete Java base WebSphere MQ , también está pensado para su uso en aplicaciones JMS WebSphere MQ y es adecuado para su uso en entornos Java Platform, Enterprise Edition (Java EE).

Una forma típica en la que puede utilizar el paquete de cabeceras WebSphere MQ es manipular mensajes de administración en formato PCF (Programmable Command Format), por ejemplo, por alguna de las razones siguientes:

- Para acceder a los detalles sobre un recurso de WebSphere MQ .
- Para supervisar la profundidad de una cola.
- Para inhibir el acceso a una cola.

Mediante el uso de mensajes PCF con la API JMS de WebSphere MQ , este tipo de administración de recursos centrados en aplicaciones se puede realizar desde aplicaciones Java EE sin tener que recurrir a la API Java base de WebSphere MQ .

Procedimiento

- Para utilizar el paquete de cabeceras WebSphere MQ para manipular cabeceras de mensajes para clases WebSphere MQ para Java, consulte “Utilización con clases WebSphere MQ para Java” en la página 963.
- Para utilizar el paquete de cabeceras WebSphere MQ para manipular cabeceras de mensajes para JMS, consulte “Utilización con WebSphere MQ classes for JMS” en la página 964.

Utilización con clases WebSphere MQ para Java

Las clases WebSphere MQ para aplicaciones Java suelen manipular objetos MQMessage, y las clases de soporte de cabeceras pueden interactuar directamente con estos objetos, ya que entienden de forma nativa las clases WebSphere MQ para interfaces Java.

Acerca de esta tarea

WebSphere MQ proporciona algunas aplicaciones de ejemplo que muestran cómo utilizar el paquete de cabeceras de WebSphere MQ con la API Java base de WebSphere MQ (clases de WebSphere MQ para 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 las clases WebSphere MQ para Java.

En función de la plataforma que utilice, estos ejemplos se instalan en el directorio `pcf` del directorio `samples` o `tools` de la instalación de WebSphere MQ (consulte “Directorios de instalación para WebSphere MQ classes for Java” en la página 669).

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 las clases WebSphere MQ para Java.

Conceptos relacionados

"Manejo de cabeceras de mensajes de WebSphere MQ con clases WebSphere MQ para Java" en la página 691
Se proporcionan clases Java que representan distintos tipos de cabecera de mensaje. También se proporcionan dos clases auxiliares.

"Manejo de mensajes PCF con clases WebSphere MQ para Java" en la página 697
Se proporcionan clases Java para crear y analizar mensajes estructurados por PCF, y para facilitar el envío de solicitudes PCF y la recopilación de respuestas PCF.

Utilización con WebSphere MQ classes for JMS

Para utilizar las cabeceras WebSphere MQ con las clases WebSphere MQ para JMS, realice los mismos pasos esenciales que para las clases WebSphere MQ para Java. El mensaje PCF se puede crear y la respuesta se puede analizar exactamente de la misma forma utilizando el paquete de cabeceras WebSphere MQ y el mismo código de ejemplo que para las clases WebSphere MQ para Java.

Acerca de esta tarea

Para enviar un mensaje PCF utilizando la API WebSphere MQ , la carga útil del mensaje se debe escribir en un mensaje de bytes JMS y se debe enviar utilizando las API JMS estándar. La única consideración es que el mensaje no debe contener un JMS RFH2 ni ninguna otra cabecera con valores específicos en MQMD.

Para enviar un mensaje PCF, siga los pasos siguientes. La forma en que se crea el mensaje PCF y se extrae información del mensaje de respuesta es la misma que para las clases WebSphere MQ para Java (consulte "Utilización con clases WebSphere MQ para Java" en la página 963).

Procedimiento

1. Cree un destino de cola JMS que represente el SYSTEM.ADMIN.COMMAND.QUEUE.

   WebSphere MQ Las aplicaciones 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 utiliza 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.

   Es necesario crear un mensaje de bytes JMS y escribir en él el mensaje PCF. Hay que crear una cola de respuestas, pero no es necesario que tenga una configuración concreta.

   El siguiente fragmento de código de ejemplo muestra cómo crear un mensaje de bytes JMS y escribir un objeto com.ibm.mq.headers,pcf.PCFMessage en él. El objeto PCFMessage (pcfCmd) se ha creado anteriormente utilizando el paquete de cabeceras de WebSphere 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 JMS” en la página 825

Los mensajes JMS se componen de una cabecera, propiedades y un cuerpo. JMS define cinco tipos de cuerpo de mensaje.

Utilización de servicios web en WebSphere MQ

Puede desarrollar aplicaciones IBM WebSphere MQ para servicios web utilizando el transporte IBM WebSphere MQ para SOAP o el puente IBM WebSphere MQ para HTTP.

El transporte IBM WebSphere MQ para SOAP proporciona un transporte JMS para SOAP. El transporte de IBM WebSphere MQ para SOAP también se integra en otros entornos como, por ejemplo, Microsoft Windows Communication Foundation, WebSphere Application Server y CICS Transaction Server.

Para obtener más información sobre el transporte IBM WebSphere MQ para SOAP, consulte “Transporte de WebSphere MQ para SOAP” en la página 966.

Con el puente IBM WebSphere MQ para HTTP, las aplicaciones cliente pueden intercambiar mensajes con IBM WebSphere MQ sin necesidad de instalar un cliente MQI de WebSphere MQ . Puede llamar a WebSphere MQ desde cualquier plataforma o idioma con prestaciones HTTP.

Para obtener más información sobre el puente IBM WebSphere MQ para HTTP, consulte “Puente WebSphere MQ para HTTP” en la página 1042.

Conceptos relacionados

“Conceptos de desarrollo de aplicaciones” en la página 8

Puede utilizar una elección de lenguajes procedimental u orientados a objetos para escribir aplicaciones de IBM WebSphere MQ. Utilice los enlaces de este tema para obtener información sobre los conceptos de IBM WebSphere MQ que son útiles para los desarrolladores de aplicaciones.

"Decidir qué lenguaje de programación utilizar" en la página 80
Utilice esta información para obtener información sobre los lenguajes de programación y las infraestructuras soportadas por IBM WebSphere MQ, y algunas consideraciones para utilizarlos.

"Diseño de aplicaciones IBM WebSphere MQ" en la página 91
Cuando haya decidido cómo pueden beneficiarse las aplicaciones de las plataformas y entornos disponibles, tendrá que decidir cómo utilizar las características que ofrece WebSphere MQ.

"Programas WebSphere MQ de ejemplo" en la página 98
Utilice esta colección de temas para obtener información sobre programas WebSphere MQ de ejemplo en distintas plataformas.

"Escritura de una aplicación de gestión de colas" en la página 199
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 cliente" en la página 359
Lo que necesita saber para escribir aplicaciones cliente en WebSphere MQ.

"Escritura de aplicaciones de publicación/suscripción" en la página 284
Empiece a escribir aplicaciones de publicación/suscripción de WebSphere MQ .

"Creación de una aplicación IBM WebSphere MQ" en la página 438
Utilice esta información para aprender a crear una aplicación IBM WebSphere MQ en distintas plataformas.

"Manejo de errores de programa" en la página 559
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.

Transporte de WebSphere MQ para SOAP

El transporte WebSphere MQ para SOAP proporciona un transporte JMS para SOAP. El transporte de WebSphere MQ para SOAP también se integra en otros entornos como Microsoft Windows Communication Foundation, WebSphere Application Server y CICS Transaction Server.

Introducción al transporte IBM WebSphere MQ para SOAP

El transporte IBM WebSphere MQ para SOAP proporciona un transporte JMS para SOAP. El emisor y escucha SOAP de WebSphere MQ proporciona un medio para llamar a los servicios web.

El escucha SOAP de WebSphere MQ da soporte a servicios alojados por .NET Framework 1, .NET Framework 2 y Axis 1.4. El remitente SOAP de WebSphere MQ soporta clientes de servicios web que se ejecutan en .NET Framework 1, .NET Framework 2, Axis 1.4 y Axis2. Los clientes pueden ser un servidor WebSphere MQ o una aplicación cliente. El transporte IBM WebSphere MQ para SOAP también se integra en otros entornos como Microsoft Windows Communication Foundation, WebSphere Application Server y CICS Transaction Server.

La integración en Microsoft Windows Communication Foundation forma parte del soporte de IBM WebSphere MQ para .NET Framework 3.

El transporte IBM WebSphere MQ para SOAP es un conjunto de protocolos y herramientas para transportar mensajes SOAP utilizando JMS sobre IBM WebSphere MQ. Se empaqueta de distintas formas para distintos entornos de aplicación, tal como se muestra en la Tabla 137 en la página 967.

Tabla 137. Transporte de IBM WebSphere MQ para entornos de aplicación SOAP		
	Integrado con componentes adicionales de WebSphere MQ	**Integrado en un marco**
Se proporciona como parte de la instalación de WebSphere MQ	.NET Framework 1 .NET Framework 2 Eje 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (solo cliente)
Proporcionado en otro paquete de software		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

La integración del transporte IBM WebSphere MQ para SOAP en una infraestructura de aplicaciones simplifica el desarrollo y el despliegue de servicios web en IBM WebSphere MQ.

Con componentes SOAP de IBM WebSphere MQ adicionales, puede interactuar directamente con los componentes SOAP de WebSphere MQ para desarrollar y desplegar servicios. Utilice las herramientas SOAP de IBM WebSphere MQ para configurar y desplegar los servicios web y los clientes de servicios web en IBM WebSphere MQ.

En los entornos integrados, el desarrollo y el despliegue son más sencillos. Puede utilizar las mismas herramientas para el desarrollo y el despliegue que para desarrollar y desplegar un servicio web HTTP SOAP. Debe seguir configurando las colas, canales y gestores de colas de IBM WebSphere MQ que necesite utilizando las herramientas de WebSphere MQ .

Puede combinar y comparar clientes y servidores SOAP de IBM WebSphere MQ desde cualquiera de estos entornos.

Ventajas

El transporte WebSphere MQ para SOAP ofrece a los usuarios IBM WebSphere MQ existentes las siguientes ventajas principales:

Utilización de la red de IBM WebSphere MQ para conectar servicios web existentes.

Los servicios pueden ser los que ha escrito o los que se proporcionan como interfaces para otras aplicaciones de software empaquetadas que ha desplegado.

La ventaja se obtiene al utilizar la red existente de WebSphere MQ para conectar servicios web. El transporte de IBM WebSphere MQ tiene la ventaja de ser un servicio de mensajería en cola gestionado y fiable.

Escribir nuevas aplicaciones, o convertir aplicaciones existentes, para utilizar SOAP en lugar de interfaces IBM WebSphere MQ .

Normalmente, las aplicaciones requieren que se desarrolle un adaptador WebSphere MQ específico para integrarse con otra aplicación. Los adaptadores tienen dos partes: la pieza de conector, que coloca y obtiene mensajes hacia y desde el transporte, y la pieza de adaptador que convierte datos a y desde formatos específicos de la aplicación. Integrar cada par de aplicaciones es un nuevo reto.

La ventaja de SOAP proviene de la estandarización en SOAP para definir interfaces de aplicación y, a continuación, tener una opción de transportes. No es necesario escribir adaptadores específicos de la aplicación y puede elegir si desea utilizar IBM WebSphere MQ o HTTP como conector. El transporte que elija dependerá de las calidades de servicio y conectividad que necesite.

Para los usuarios de SOAP sobre HTTP existentes, la ventaja del transporte WebSphere MQ para SOAP proviene del uso de un transporte asíncrono gestionado y fiable. Los beneficios son dobles:

Un modelo de programación realmente asíncrono para la disponibilidad y el rendimiento.
Al utilizar una interfaz de cliente asíncrona, las aplicaciones de cliente y servicio no tienen que estar disponibles al mismo tiempo. Las solicitudes enviadas por el cliente se almacenarán hasta que el servicio esté disponible para procesarlas.

Una red gestionada preparada que está diseñada para ser fiable y disponible.

Al elegir IBM WebSphere MQ como transporte, obtiene la ventaja de utilizar una red gestionada que proporciona mensajería fiable.

Por el contrario, los transportes como HTTP y FTP sobre TCP/IP no están gestionados. Una red no gestionada es ideal para conexiones imprevisibles: hay menos tareas de gestión.

Resumen

El transporte IBM WebSphere MQ para SOAP proporciona los componentes siguientes:

- El enlace de transporte SOAP/JMS se utiliza en documentos WSDL para enlazar un servicio SOAP a un transporte JMS. La implementación de WebSphere MQ del enlace SOAP/JMS utiliza un URI que tiene dos formatos:

  Transporte de WebSphere MQ para SOAP

  ```
  jms:/queue?&Name=Value&Name=Value...
  ```

  Formato físico de WebSphere MQ para la recomendación de candidato W3C

  ```
  jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
  ```

- Correlación de un mensaje SOAP con un mensaje WebSphere MQ .
- Dos escuchas SOAP de IBM WebSphere MQ para recibir solicitudes SOAP, uno para Java y otro para .NET Framework 1 o .NET Framework 2. Los escuchas utilizan .NET o Axis 1.4 para procesar la solicitud SOAP.
- Dos remitentes SOAP de IBM WebSphere MQ para crear solicitudes SOAP de IBM WebSphere MQ . clientes de servicios web se registran con un remitente para procesar solicitudes SOAP de `jms:` .
- Integración con Windows Communication Foundation (WCF), a veces conocido como .NET 3, para enviar y recibir mensajes WebSphere MQ Transport para SOAP.
- Integración del cliente con Axis2, a veces conocido como JAX-WS, para enviar mensajes JMS SOAP de WebSphere MQ Transport for SOAP o W3C .
- El mandato **amqwdeployWMQService**, que crea componentes y scripts de desarrollo y tiempo de ejecución para desplegar un servicio web utilizando el transporte IBM WebSphere MQ para SOAP.
- Código de servicio y cliente Java y .NET de ejemplo.
- Un script para establecer la vía de acceso de clases y otros scripts de programa de utilidad.

En los entornos integrados, el emisor y el escucha se integran en cada entorno, al igual que las extensiones de las herramientas de desarrollo y despliegue.

Integración de SOAP y WebSphere MQ

El transporte de WebSphere MQ para SOAP amplía las herramientas SOAP y de servicios web y el tiempo de ejecución, con WebSphere MQ como transporte alternativo a HTTP para SOAP. No es necesario modificar los servicios web existentes para utilizar el transporte WebSphere MQ para SOAP como transporte. El transporte utiliza un formato de URI personalizado para SOAP/JMS. El formato de URI W3C para SOAP/JMS está soportado de forma limitada por los clientes Axis2 .

Se debe añadir una línea de código adicional a los clientes en los entornos .NET Framework 1, .NET Framework 2 y Axis 1.4 . No es necesario ningún código adicional en los clientes Axis 2 y Windows Communication Foundation (WCF). El escucha SOAP de WebSphere MQ ejecuta servicios en los entornos .NET Framework 1, .NET Framework 2 y Axis 1.4 . El transporte de WebSphere MQ para SOAP está integrado en otros entornos de servidor de aplicaciones, incluidos WCF, CICS y WebSphere Application Server.

¿Qué es SOAP?

SOAP[9]describe el formato estandarizado de los mensajes y protocolos de interacción que las aplicaciones utilizan para intercambiar solicitudes, respuestas y datagramas. SOAP es independiente del transporte utilizado para transferir los mensajes y del entorno de aplicación que envía y recibe los mensajes. El W3C define SOAP Versión 1.2 sucintamente:

SOAP Versión 1.2 proporciona la definición de la información basada en XML que se puede utilizar para intercambiar información estructurada y escrita entre iguales en un entorno distribuido descentralizado.[10].

Para utilizar SOAP, debe estar enlazado a un transporte, como HTTP, correo electrónico o WebSphere MQ.

Una infraestructura de enlace de protocolo SOAP es el conjunto de reglas para transportar un mensaje SOAP encima de otro protocolo, como HTTP. Versión de SOAP 1.2 Parte 2: Adjuntos (Segunda edición) describe el enlace HTTP SOAP.

La recomendación de candidato W3C , 4 de junio de 2009, SOAP sobre Java Message Service 1.0, describe la recomendación para el enlace JMS SOAP. Como JMS es una especificación de API, y no un protocolo de transporte, la recomendación SOAP de JMS no describe el formato físico de los mensajes JMS de SOAP. Describe los protocolos de interacción SOAP y el enlace de API JMS. En consecuencia, al utilizar la recomendación SOAP de JMS, debe seguir utilizando la misma implementación JMS para el cliente SOAP y el servidor SOAP. Permite que una aplicación JMS SOAP se ejecute en cualquier implementación de JMS. Una implementación JMS se puede conectar a un servidor de aplicaciones J2EE , si tanto el servidor como la implementación JMS cumplen con la especificación JCA. WebSphere MQ JMS cumple con la especificación JCA y se puede conectar a un servidor de aplicaciones compatible.

El transporte WebSphere MQ para el enlace SOAP es como el estándar W3C propuesto, pero no es el mismo. Su uso se describe en el tema MQRFH2 Valores SOAP. A diferencia de la recomendación de candidato W3C , el enlace SOAP no se especifica formalmente. De hecho, es el enlace HTTP y la dirección de servicio toma el formato `jms:/queue?name=value&name=value...`, en lugar de `http://authority/path?query#fragment`. `jms:` no es un esquema de URI de IANA registrado oficialmente.

¿Qué es un servicio web?

SOAP permite que los programas escritos en distintos lenguajes, que se ejecutan en distintas plataformas, se comuniquen utilizando diversos protocolos de transporte. SOAP es la especificación de protocolo. Un servicio web es una aplicación que proporciona un servicio a través de una interfaz SOAP a la que se puede acceder utilizando protocolos de Internet.

Un objetivo importante de SOAP es proporcionar servicios que los clientes pueden utilizar fácilmente. Una vez que haya diseñado un cliente para utilizar un servicio, puede programar la llamada para invocar el servicio sin hacer referencia a la documentación externa. Las interfaces de servicio se describen en XML, en un documento WSDL. La consulta, `http://authority/path?wsdl`, devuelve la descripción WSDL de un servicio SOAP.

Consejo: Cuando despliegue un servicio web para utilizar WebSphere MQ, despliegue también el servicio en HTTP para que funcione la consulta WSDL estándar.

Desarrollo de servicios web

Los servicios web tienen un cliente y un componente de servicio. El servicio se escribe primero, empezando por la descripción de interfaz en WSDL, o siguiendo las reglas para escribir la clase de servicio. Los kits de herramientas de servicio web tienen programas de utilidad para generar WSDL a partir de la definición de interfaz de una clase; por ejemplo, **`java2wsdl`** o **`disco`**. También tienen herramientas para generar o clasificar esqueletos a partir de descripciones de interfaz WSDL; por ejemplo, **`wsdl2java`**, **`wsimport`**o **`wsdl`**. El primero se conoce como desarrollo de abajo hacia arriba, y el segundo de arriba hacia abajo.

---

[9] Históricamente, el acrónimo representaba Simple Object Access Protocol.

[10] W3C: SOAP Version 1.2 Parte 0

El mandato **amqwdeployWMQService** en WebSphere MQ Transport para SOAP utiliza estas herramientas para generar WSDL, apéndices de cliente y proxies de cliente.

Los servicios web normalmente se escriben utilizando un entorno de desarrollo integrado destinado a un entorno de servidor de aplicaciones determinado:

Eclipse IDE para desarrolladores Java EE
Crea servicios web para el eje 2. Da soporte a JAX-RPC y JAX-WS

Rational Application Developer V7.5
Crea servicios web para WebSphere Application Server V7 y versiones anteriores, y también para Axis. Soporta JAX-RPC y JAX-WS.

WebSphere Integration Developer V6.2
Crea servicios web para WebSphere Process Server y WebSphere ESB. Soporta JAX-RPC y JAX-WS.

Visual Studio 2008 (Versión 9)
Crea servicios web para .NET Framework 3.5 y anteriores (Windows Communication Foundation)

Visual Studio 2005 (Versión 8)
Crea servicios web para .NET Framework 2 y anteriores

Puede utilizar cualquiera de estas herramientas en combinación con el transporte WebSphere MQ para SOAP. Una vez que haya desarrollado un servicio para utilizarlo con HTTP, utilice la herramienta **amqwdeployWMQService** para desplegar los servicios para utilizar WebSphere MQ como transporte. Puede escribir un nuevo cliente utilizando la salida de la herramienta, o modificar los clientes existentes para utilizar el transporte WebSphere MQ para SOAP.

Si el transporte WebSphere MQ para SOAP está integrado en el entorno de aplicación, no es necesario utilizar la herramienta **amqwdeployWMQService** ni modificar el código de cliente. La capa SOAP de cliente dirige las solicitudes de cliente que tienen un URI con el prefijo `jms:` al transporte WebSphere MQ para SOAP. La capa SOAP de servidor llama al transporte WebSphere MQ para que SOAP espere a las solicitudes SOAP de `jms:` y devuelve respuestas al transporte WebSphere MQ para SOAP.

Normalmente, los servicios .NET se han desarrollado de abajo a arriba utilizando anotaciones de servicio web en código y servicios Java de arriba abajo, utilizando definiciones de interfaz WSDL. La diferencia en los enfoques se reduce, ya que Java Standard Edition Versión 6 da soporte a JAX-WS 2.0, y utiliza anotaciones para calificar la definición de interfaces de servicio. Ahora es tan fácil desarrollar los servicios Java de abajo hacia arriba como de arriba hacia abajo. El enfoque que elija es una cuestión de método de desarrollo.

El cliente de servicios web se escribe después del servicio, utilizando la definición de servicio WSDL y los apéndices y proxies de cliente generados. En algunas aplicaciones, la definición de servicio no se conoce cuando se escribe el cliente. El cliente recupera el WSDL de servicio y crea solicitudes de servicio dinámicamente. Más comúnmente, se conoce la definición de servicio, pero la dirección en la que se despliega el servicio no lo es. El kit de herramientas de servicio web genera interfaces para que el cliente las utilice para realizar solicitudes de servicio. El cliente proporciona la dirección de servicio cuando es necesario. En el tercer caso, el WSDL contiene toda la información que necesita un cliente. El WSDL contiene la interfaz y la dirección del servicio. El código generado por el kit de herramientas de servicio web tiene toda la información que necesita el cliente para realizar solicitudes de un servicio.

Puede utilizar cualquiera de estos tres estilos con el transporte WebSphere MQ para SOAP.

Entornos de aplicación de servicio web

Los kits de herramientas de servicio web requieren una correlación de la definición WSDL de un servicio con las corrientes de bytes que se transfieren en solicitudes y respuestas SOAP. La corriente de bytes está definida por la especificación SOAP y está contenida en el sobre SOAP. El sobre SOAP se muestra en Figura 166 en la página 971.

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
<!-- headers... -->
</soap:Header>
<soap:Body>
<!-- payload or fault message -->
</soap:Body>
</soap:Envelope>
```

Figura 166. sobre SOAP

La correlación del sobre SOAP con el enlace de lenguaje y viceversa es parte estandarizada y parte propietaria. La correlación es fundamental para la arquitectura .NET y se proporciona como parte de Common Language Runtime (CLR). La correlación se estandariza en Java mediante especificaciones JAX. Debido a que las correlaciones Java están estandarizadas, los clientes y servicios de servicio web Java son portables entre distintos entornos de aplicación basados en Java. JAX-RPC (a veces denominado JAX-WS 1.0) es la correlación que más se utiliza hoy en día. Está soportado por Axis 1.4. JAX-WS (a veces denominado JAX-WS 2.0) es un estándar muy mejorado y es probable que sustituya JAX-RPC rápidamente. JAX-WS está soportado por Axis 2.0. WebSphere MQ 7.0.1 no da soporte a JAX-WS y Axis 2.

El transporte WebSphere MQ para SOAP no altera el contenido del sobre SOAP y el contenido no afecta al transporte. Los enlaces de lenguaje afectan al transporte de WebSphere MQ para SOAP. WebSphere MQ 7.0.1 da soporte a .NET Framework 1, .NET Framework 2 y Axis 1.4 utilizando el código y los programas de utilidad que se proporcionan con el transporte WebSphere MQ para SOAP. El soporte para el transporte WebSphere para SOAP en .NET Framework 3 y 3.5 se implementa utilizando el canal personalizado WebSphere MQ para Windows Communication Foundation.

Otros entornos de desarrollo y tiempo de ejecución SOAP podrían proporcionar soporte para el transporte de WebSphere MQ para SOAP y dar soporte a distintos lenguajes. Por ejemplo, los servicios web que se ejecutan en CICS dan soporte a lenguajes como COBOL y PL/1.

Nota: La correlación utilizada no hace ninguna diferencia en la interoperatividad de los servicios web. Puede combinar y comparar clientes y servicios escritos utilizando correlaciones .NET, JAX-RPC y JAX-WS.

¿Qué es el transporte WebSphere MQ para SOAP?

WebSphere MQ Transport para SOAP es un enlace SOAP y un kit de herramientas de servicios web. Juntos, permiten que las aplicaciones intercambien mensajes SOAP utilizando WebSphere MQ en lugar de HTTP. Figura 167 en la página 971 muestra WebSphere MQ como una alternativa a HTTP como un transporte SOAP.

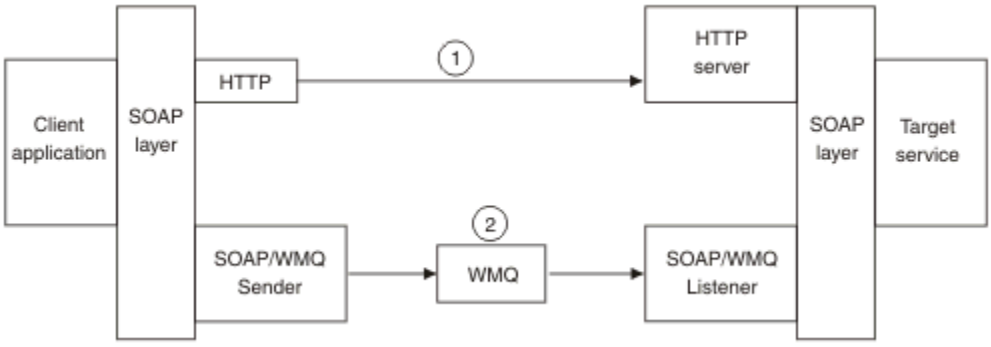


Figura 167. Visión general del transporte WebSphere MQ para SOAP

SOAP sobre HTTP se muestra como (1) en el diagrama. La capa SOAP de cliente convierte una solicitud en un mensaje SOAP y el componente HTTP envía a través de TCP/IP. El componente de servidor HTTP está a la escucha de solicitudes HTTP, normalmente en el puerto TCP/IP 80. Si la solicitud es para un servicio SOAP, el componente de servidor HTTP llama a la capa SOAP para convertir la solicitud SOAP en una llamada de método. A continuación, devuelve la respuesta.

SOAP sobre WebSphere MQ se muestra como (2). La aplicación cliente registra el componente emisor SOAP WebSphere MQ como un manejador para el protocolo `jms:` con la capa SOAP. La capa SOAP pasa mensajes SOAP dirigidos a `jms:` al remitente SOAP de WebSphere MQ . El emisor utiliza el URI en el mensaje para colocar el mensaje en la cola de solicitudes con las calidades de servicio necesarias. El escucha SOAP de WebSphere MQ correspondiente espera mensajes en su cola de solicitudes y llama a la capa SOAP para procesar solicitudes y devolver respuestas.

El emisor y el escucha SOAP son programas WebSphere MQ normales. Se pueden conectar al mismo gestor de colas, como en Figura 168 en la página 972, o se pueden conectar a distintos gestores de colas; consulte Figura 169 en la página 973. El cliente se puede conectar mediante una conexión de cliente.

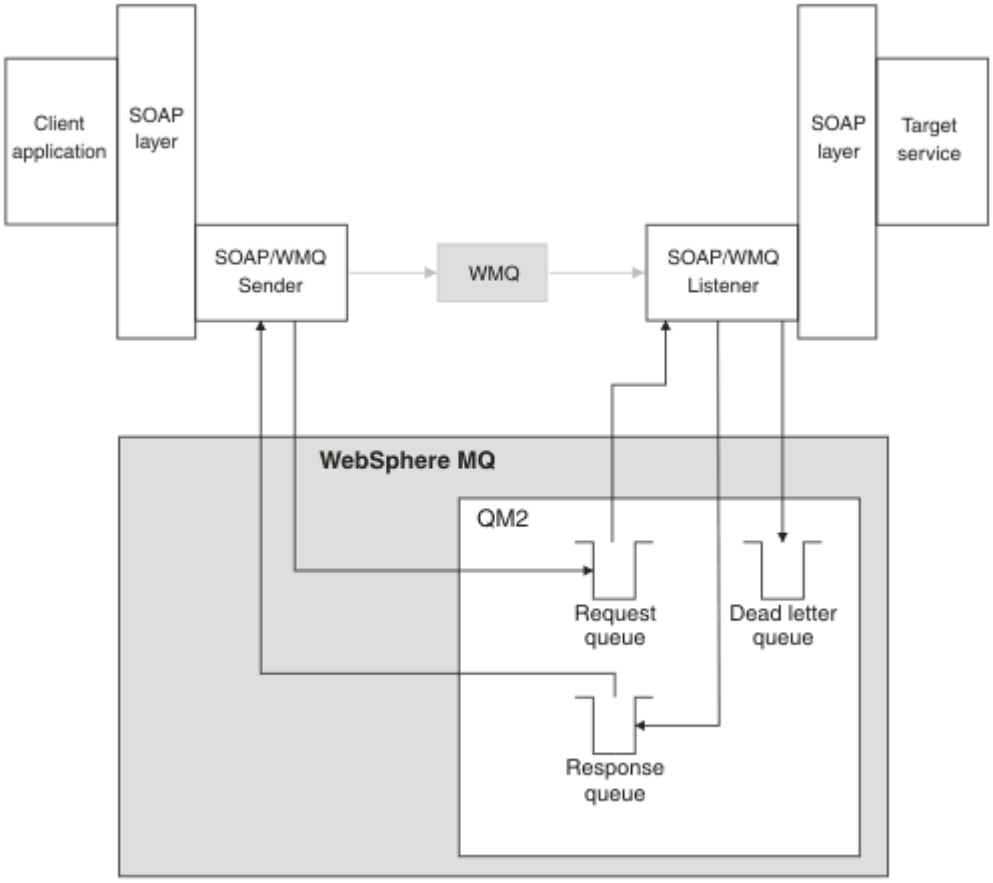


Figura 168. Colas utilizadas por SOAP/WebSphere MQ (gestor de colas único)

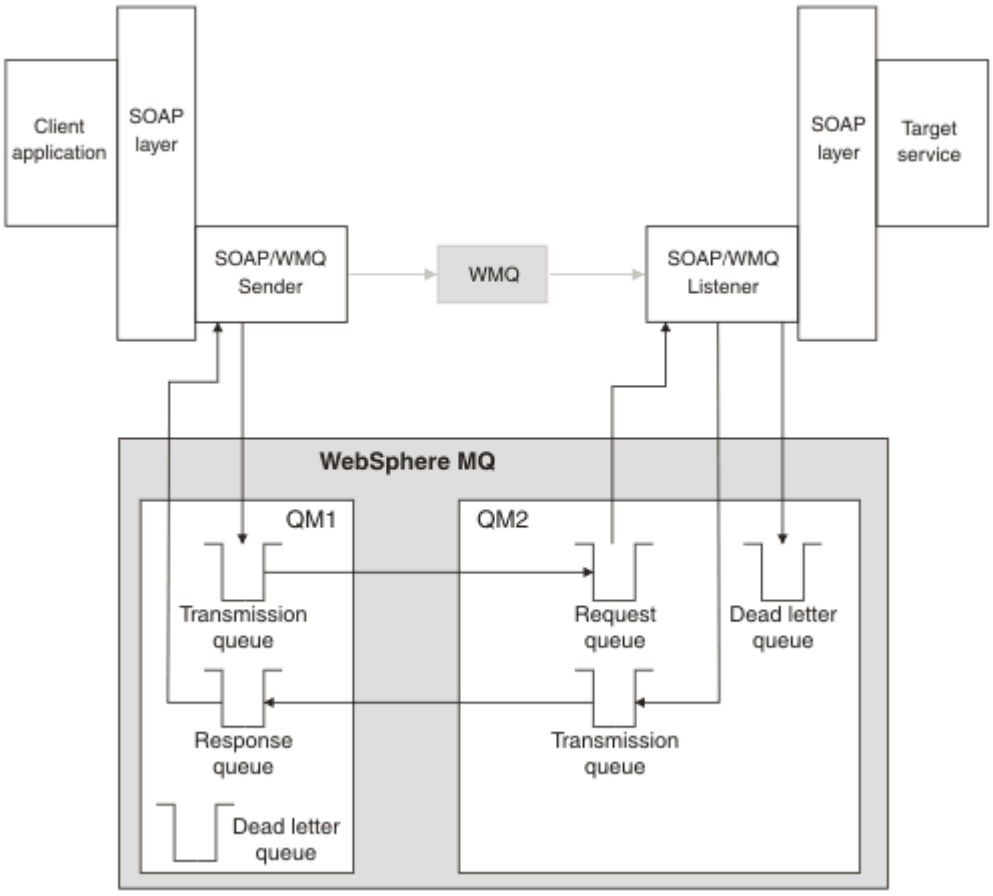


Figura 169. Colas utilizadas por SOAP/WebSphere MQ (gestores de colas independientes)

Recomendación de candidato W3C para enlazar SOAP con JMS.

La recomendación candidata de W3C define el enlace SOAP sobre JMS; SOAP sobre Java Message Service 1.0. También es útil para sus ejemplos URI Scheme for Java (tm) Message Service 1.0[11].

Algunas infraestructuras de aplicación, como por ejemplo WebSphere Application Server v7, tienen soporte para la recomendación de candidato W3C . Envíe solicitudes SOAP formateadas con un URI compatible con la recomendación candidata W3C utilizando el cliente Axis2 ; consulte W3C URI SOAP sobre JMS para el cliente WebSphere MQ Axis 2 . El cliente Axis2 envía una solicitud SOAP formateada con un W3C o un transporte WebSphere MQ para SOAP basado en URI en la solicitud SOAP.

El soporte de cliente Axis2 para la recomendación W3C se introduce en el fixpack 7.0.1.3 . No se proporciona soporte para otros clientes y para los escuchas SOAP proporcionados por WebSphere MQ .

Conceptos relacionados

Implementación del transporte WebSphere para SOAP en .NET Framework 1, .NET 2 y Axis 1.4
Es posible que desee escribir su propio emisor y escucha SOAP de WebSphere MQ . Utilice la implementación de WebSphere MQ Transport para SOAP en .NET Framework 1, .NET Framework 2 y Axis 1.4 como guía.

Transporte de WebSphere MQ para mensajería fiable de servicios SOAP y web
La mensajería fiable de servicios web es un protocolo para intercambiar de forma fiable solicitudes y respuestas de servicios web a través de una conexión no fiable. Es el más adecuado para resolver problemas de interrupción de la conexión de corta duración.

[11] Busque *URI Scheme for JMS*, en las referencias de especificación de W3C, para obtener el borrador más reciente.

Implementación del transporte WebSphere para SOAP en .NET Framework 1, .NET 2 y Axis 1.4

Es posible que desee escribir su propio emisor y escucha SOAP de WebSphere MQ . Utilice la implementación de WebSphere MQ Transport para SOAP en .NET Framework 1, .NET Framework 2 y Axis 1.4 como guía.

1. Un programa cliente utiliza la infraestructura de servicios web adecuada de la misma forma que lo haría para el transporte HTTP. También debe registrar el prefijo `jms:` . El prefijo se registra utilizando el método Java `com.ibm.mq.soap.Register.extension()` o el método CLR `IBM.WMQSOAP.Register.Extension()` .
2. La infraestructura Axis 1.4 o .NET Framework 1 o 2 ordena la llamada en un mensaje de solicitud SOAP exactamente igual que para SOAP/HTTP.
3. Un servicio de WebSphere MQ se identifica mediante un URI con el prefijo `jms:`. Cuando la infraestructura identifica el URI `jms:` , llama al código del remitente de transporte de WebSphere MQ ; `com.ibm.mq.soap.transport.jms.WMQSender` (para Axis 1.4) o `IBM.WMQSOAP.MQWebRequest` (para .NET1 y 2). Si la infraestructura encuentra un URI con un prefijo `http:` , llama al remitente SOAP sobre HTTP estándar.
4. El mensaje SOAP es transportado por el remitente SOAP de WebSphere MQ utilizando la cola de solicitudes. **`SimpleJavaListener`** (para Java) o **`amqwSOAPNETListener`** (para .NET) recibe el mensaje de solicitud.

   Los escuchas SOAP de WebSphere MQ son procesos autónomos y tienen varias hebras con un número personalizable de hebras.
5. El escucha SOAP de WebSphere MQ lee la solicitud SOAP de entrada y la pasa a la infraestructura de servicio web adecuada.
6. La infraestructura de servicio web analiza el mensaje de solicitud SOAP e invoca el servicio. El procedimiento es el mismo que para un mensaje que ha llegado a un transporte HTTP.
7. La infraestructura formatea la respuesta en un mensaje de respuesta SOAP y la devuelve al escucha SOAP de WebSphere MQ .
8. El escucha coloca el mensaje en la cola de respuestas y el mensaje se transfiere al remitente SOAP de WebSphere MQ . El remitente lo pasa a la infraestructura de servicio web del cliente.
9. La infraestructura de cliente analiza el mensaje SOAP de respuesta y devuelve el resultado a la aplicación cliente.

Cada contexto de aplicación es servido por una cola de solicitudes de WebSphere MQ independiente.

El contexto de aplicación se controla en Axis 1.4 asegurándose de que el servicio y el escucha SOAP de WebSphere MQ se ejecutan en el directorio adecuado. El eje 1.4 establece la CLASSPATH correcta para el directorio.

El contexto de aplicación está controlado en .NET por el escucha SOAP de WebSphere MQ que ejecuta el servicio en un contexto creado por una llamada a `ApplicationHost.CreateApplicationHost`. La llamada especifica el directorio de ejecución de destino. A continuación, cada servicio opera en el directorio en el que se ha desplegado.

`amqwdeployWMQService` genera las colas de solicitud y respuesta. También genera la infraestructura necesaria para manejar las colas y desplegar servicios en Axis 1.4.

Conceptos relacionados

Integración de SOAP y WebSphere MQ

Transporte de WebSphere MQ para mensajería fiable de servicios SOAP y web

La mensajería fiable de servicios web es un protocolo para intercambiar de forma fiable solicitudes y respuestas de servicios web a través de una conexión no fiable. Es el más adecuado para resolver problemas de interrupción de la conexión de corta duración.

Transporte de WebSphere MQ para mensajería fiable de servicios SOAP y web

La mensajería fiable de servicios web es un protocolo para intercambiar de forma fiable solicitudes y respuestas de servicios web a través de una conexión no fiable. Es el más adecuado para resolver problemas de interrupción de la conexión de corta duración.

WebSphere MQ para SOAP aprovecha la utilización de una red gestionada y fiable de WebSphere MQ para pasar mensajes SOAP. Los transportes como HTTP y FTP no están gestionados. Las redes no gestionadas son ideales para conexiones impredecibles, donde las dificultades y los costes de gestionar las conexiones superan los beneficios de no perder solicitudes y respuestas.

Para superar el problema de la pérdida de archivos cuando las conexiones se rompen en redes no gestionadas, los servicios como FTP gestionado construyen una capa de gestión sobre FTP. La capa de gestión se hace cargo de la carga de comprobar que los archivos se han transferido correctamente de los usuarios y retransmite los archivos que faltan si es necesario. Para utilizar FTP gestionado, debe tener el software de gestión instalado en ambos extremos de la conexión.

La mensajería fiable de servicios web (WSRM) adopta un enfoque diferente para resolver el problema de las conexiones no fiables. Su objetivo es transferir solicitudes y respuestas de servicio web de forma fiable, sin que ambos extremos de la conexión tengan que utilizar el mismo software. Cualquier software, mediante la implementación del protocolo de mensajería fiable de servicios web, puede intercambiar mensajes de forma fiable con otro software.

Cuando falla una conexión, un emisor y un receptor deben conservar el contexto de la transferencia de mensajes WSRM, utilizando un URI generado como clave. El remitente y el destinatario siguen intentando establecer una nueva conexión. Si se establece correctamente una nueva conexión, la transferencia se completa. La especificación WSRM no especifica cómo se conserva el contexto o cuándo se intenta una nueva conexión.

Usted podría decidir que sólo las interrupciones de corta duración son de interés. Para paradas más largas, es posible que esté preparado para descartar transferencias que no se han podido reiniciar después de un tiempo. De forma similar, es posible que esté preparado para descartar transferencias si falla el cliente o el servicio. Dejando al usuario responsable de asegurar las transferencias, pone menos demandas en la gestión de la coordinación de cliente y servicio.

Si las interrupciones de la red son de larga duración, de más de 30 minutos aproximadamente, o si el cliente o el servidor falla, existe una mayor probabilidad de que algunas conexiones nunca se restablezcan. Ya no puede confiar en que WSRM restaure la transferencia de mensajes automáticamente de forma no gestionada. Debe considerar la posibilidad de gestionar las conexiones WSRM fallidas, lo que implica desarrollar software para gestionar la red de clientes y servicios.

El uso de WSRM para superar paradas cortas puede reducir significativamente el manejo de mensajes perdidos en una red móvil. Si no tiene que garantizar la entrega de mensajes, las ventajas de reducir la pérdida de mensajes podrían justificar el coste adicional de desarrollar una implementación de WSRM.

SOAP sobre JMS proporciona una entrega de mensajes asegurada y se ocupa de las interrupciones de mayor duración del cliente, el servidor y la red. Si está buscando una calidad de servicio más fiable para SOAP que HTTP, ¿qué solución elige: WebSphere MQ transport for SOAP o WSRM? La respuesta depende de muchos factores. Algunos de los factores a considerar se enumeran:

1. Si la falta de fiabilidad se debe a un error de conexión.
2. Cuánto tiempo amados son los fallos de conexión.
3. Si puede gestionar tanto el cliente como el servidor de la conexión.
4. Si el usuario o un administrador es en última instancia responsable de la entrega de mensajes.

Conceptos relacionados

Integración de SOAP y WebSphere MQ

Implementación del transporte WebSphere para SOAP en .NET Framework 1, .NET 2 y Axis 1.4
Es posible que desee escribir su propio emisor y escucha SOAP de WebSphere MQ . Utilice la implementación de WebSphere MQ Transport para SOAP en .NET Framework 1, .NET Framework 2 y Axis 1.4 como guía.

Instalación y verificación de servicios web de WebSphere MQ

Utilice las instrucciones de estos temas para instalar y verificar el transporte WebSphere MQ para SOAP.

Instalación del transporte web de WebSphere MQ para SOAP

Utilice estas instrucciones para instalar el transporte web de WebSphere MQ para SOAP. La instalación crea herramientas para ejecutar clientes o servicios de servicio web utilizando WebSphere MQ como transporte SOAP. Las herramientas se utilizan en los entornos SOAP de .NET Framework 1, .NET 2, Axis 1.4o Axis2 .

Antes de empezar

Compruebe los productos que son requisito previo en Requisitos del sistema para IBM WebSphere MQ. El proceso de instalación no comprueba la presencia ni la disponibilidad del software de requisito previo. Hay que verificar que los requisitos previos están instalados.

WebSphere MQ proporciona una copia del tiempo de ejecución de Axis 1.4 . Utilice esta versión con WebSphere MQ en lugar de cualquier otra que haya instalado. IBM no proporciona soporte técnico para Apache Axis. Póngase en contacto con la Apache Software Foundation si tiene problemas técnicos con él.

Para ejecutar servicios web en el entorno SOAP de .NET Framework 3, WebSphere MQ utiliza Windows Communication Foundation. El canal personalizado de WebSphere MQ para Windows Communication Foundation ejecuta clientes y servicios de servicio web utilizando WebSphere MQ como transporte para mensajes SOAP.

Acerca de esta tarea

Puede instalar el transporte web de WebSphere MQ para SOAP como una aplicación de cliente o servidor MQI de WebSphere MQ . Si ya ha instalado WebSphere MQ como cliente o servidor en el sistema, compruebe que ha instalado los componentes listados.

MQ_INSTALLATION_PATH representa el directorio de alto nivel en el que está instalado WebSphere MQ.

Lleve a cabo los pasos de instalación siguientes.

Procedimiento

1. Seleccione el componente "Java and. Net Messaging and Web services" para la instalación.
2. En Solaris y HP-UX, seleccione el componente "Entorno de ejecución Java" para la instalación.
3. Seleccione el Kit de herramientas de desarrollo para la instalación.
4. Instale y verifique WebSphere MQ tal como se describe en la Guía de iniciación rápida para su plataforma.
5. Copie el tiempo de ejecución de Apache Axis 1.4 , `axis.jar` desde el directorio `prereqs/axis` en el soporte de instalación de WebSphere MQ . Cópielo en el directorio de instalación que se describe en Tabla 138 en la página 977, Tabla 139 en la página 977 o Tabla 140 en la página 977.

   Windows

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

   AIX

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

   Directorios de instalación de HP-UX, Solaris y Linux (todas las plataformas)

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

6. En Windows 2003, ejecute **Aspnet_regiis.exe** para actualizar las correlaciones de scripts para que apunten a la versión del tiempo de ejecución del lenguaje común que está utilizando.

   Busque el programa de utilidad **Aspnet_regiis.exe** en %SystemRoot%Microsoft.NET/Framework/*version-number* .
7. Establezca la variable de entorno, WMQSOAP_HOME , para que apunte al directorio de instalación de WebSphere MQ .

Resultados

Tabla 138. Directorios de instalación de Windows	
Ubicación	**Contenido**
MQ_INSTALLATION_PATH\programs\bin	Archivos binarios, comandos, DLL y ejecutables
MQ_INSTALLATION_PATH\programs\java\lib	.jar files
MQ_INSTALLATION_PATH\programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH\programs\soap\samples	Ejemplos e IVT

Tabla 139. Directorios de instalación de AIX	
Ubicación	**Contenido**
MQ_INSTALLATION_PATH/bin	Scripts de shell
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar y otros archivos JAX-RPC .jar
MQ_INSTALLATION_PATH/samp/soap	Ejemplos e IVT

Tabla 140. Directorios de instalación de HP-UX, Solaris y Linux (todas las plataformas)	
Ubicación	**Contenido**
MQ_INSTALLATION_PATH/bin	Scripts de shell
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar y otros archivos JAX-RPC .jar
MQ_INSTALLATION_PATH/samp/soap	Ejemplos e IVT

Qué hacer a continuación

1. Sólo para .NET, debe registrar el transporte WebSphere MQ para archivos SOAP con la memoria caché de conjunto global. Si .NET ya está instalado al instalar WebSphere MQ, el registro se realiza automáticamente durante la instalación. Si instala .NET después de WebSphere MQ, el registro se realiza automáticamente cuando se ejecuta por primera vez la IVT.

   Puede ejecutar **amqiregisterdotnet.cmd** para realizar el registro de los ensamblados .NET. También puede ejecutar **amqiregisterdotnet.cmd** para forzar el registro en cualquier fase. Una vez hecho, este registro sobrevive a los reinicios del sistema y no suele ser necesario volver a registrar posteriormente.
2. Ejecute la prueba de verificación de la instalación, tal como se describe en "Verificación del transporte IBM WebSphere MQ para SOAP" en la página 978.

3. Si piensa desarrollar el cliente Axis2, tendrá que descargar Axis2 1.4.1 desde Apache; consulte “Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse” en la página 1001.

Verificación del transporte IBM WebSphere MQ para SOAP

Verifique el transporte IBM WebSphere MQ para SOAP utilizando el mandato **runivt** . El mandato ejecuta una serie de aplicaciones de demostración y garantiza que el entorno se ha configurado correctamente después de la instalación.

Antes de empezar

Antes de ejecutar el mandato **runivt** , asegúrese de que tiene los siguientes entornos de ejecución:

- Para ejecutar sólo en Axis: debe tener un SDK Java (dentro de SOE) disponible en el sistema. También debe incluir la ubicación de los mandatos `java.exe` y `javac.exe` en la variable de entorno **PATH** de los sistemas.
- Para ejecutar una prueba sólo en .NET (soportado sólo en Windows): debe tener un SDK Java y los compiladores y herramientas .NET en el sistema. Para ello, acceda a un indicador de mandatos de Visual Studio o al indicador de mandatos de Microsoft Windows SDK y, a continuación, añada la ubicación de los archivos `java.exe` y `javac.exe` a la variable de entorno **PATH** .
- Para ejecutar todas las pruebas disponibles: para plataformas Windows , el entorno debe estar configurado tal como se describe en la ejecución de prueba .NET. En plataformas UNIX and Linux , el entorno debe estar configurado tal como se describe en la ejecución de prueba Axis only.

Acerca de esta tarea

En lugar de ejecutar la prueba de verificación en .NET y Axis, es posible que desee ejecutar la prueba sólo en Axis, o sólo en .NET.

Si experimenta problemas con las pruebas y desea empezar de nuevo:

1. Detenga el gestor de colas `WMQSOAP.DEMO.QM` utilizando la opción `immediate` .
2. Detenga el escucha que se ha iniciado en una ventana diferente.
3. Suprima el gestor de colas.
4. Suprima el directorio de ejemplos temporales que ha creado y vuelva a empezar.

En plataformas UNIX and Linux , debe ejecutar el mandato utilizando una sesión del sistema X Windows .

El mandato **runivt** cambia el contenido del directorio `soap/samples` . Para mantener la imagen de instalación sin modificar, copie el directorio de ejemplos en una ubicación temporal y ejecute la prueba de verificación desde la ubicación temporal.

Puede ejecutar la verificación de la instalación tantas veces como desee.

Realice los pasos siguientes para verificar la instalación de IBM WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2 y Axis 1.4:

Procedimiento

1. Copie el árbol de directorios `./tools/soap/samples` en una ubicación temporal.
2. Inicie una ventana de mandatos con el directorio temporal como directorio actual.
3. Utilice el mandato **runivt** para iniciar la prueba de instalación. El script `runivt` compila una clase de prueba, el cliente de ejemplo y los servicios antes de desplegarlos y ejecutarlos. Para que la clase de prueba, el cliente de ejemplo y los servicios se ejecuten, complete los pasos de instalación descritos en Instalación de WebSphere(r) MQ Web transport for SOAP y asegúrese de que el indicador de mandatos utilizado para ejecutar el mandato `runivt` tenga establecido el entorno de ejecución necesario. Utilice cualquiera de los métodos siguientes para ejecutar el mandato **runivt** :
   - Ejecute una prueba sólo en Axis: `runivt Axis`.
   - Ejecutar una prueba sólo en .NET (soportado sólo en Windows): `runivt DotNet`.

- Ejecute todas las pruebas disponibles:`runivt`.

Para obtener más información sobre la sintaxis y los parámetros del mandato `runivt` , consulte **runivt**: IBM WebSphere MQ transport for SOAP installation verification test. Las pruebas que puede ejecutar se listan en el archivo `ivttests.txt` en Windows y `ivttests_unix.txt` en plataformas UNIX and Linux .

Referencia relacionada

runivt: prueba de verificación de instalación de transporte de WebSphere MQ para SOAP

Desarrollo de servicios web para el transporte de WebSphere MQ para SOAP

Utilice el entorno de desarrollo de servicios web normal para desarrollar servicios para su uso con el transporte WebSphere MQ para SOAP.

Antes de empezar

1. Si tiene previsto utilizar las herramientas de línea de mandatos proporcionadas con el transporte WebSphere MQ para SOAP:
   a. Cree un directorio de despliegue para el servicio.
   b. Inicie una ventana de mandatos en el directorio.
   c. Para .NET, csc.exe y wsdl.exe deben estar en la vía de acceso y ser de la misma versión de .NET Framework.
   d. Para Java,
      i) Ejecute el mandato **amqwsetcp** para configurar la vía de acceso de clases.
      ii) Un JRE de IBM y un JDK en el mismo nivel de versión deben estar en la vía de acceso actual. El nivel de versión debe ser como mínimo 5.0.
      iii) Personalice la vía de acceso de clases para incluir las ubicaciones de las bibliotecas de `.jar` adicionales y los directorios que contienen paquetes .java, incluido el servicio que está desarrollando. Coloque el directorio actual "`.`" en la vía de acceso de clases.
      iv) Cree un directorio, relativo al directorio actual de la ventana de mandatos, correspondiente al nombre de paquete del servicio que está desarrollando.
2. De forma alternativa, utilice herramientas de entorno de trabajo que den soporte al desarrollo de servicios web. Las tareas de desarrollo de ejemplo utilizan Microsoft Visual Studio 2008, Eclipse IDE para desarrolladores Java EE y WebSphere Application Server Community Edition.

Acerca de esta tarea

Los servicios web existentes no necesitan ninguna modificación para trabajar con el transporte de WebSphere para SOAP. Las herramientas proporcionadas con el transporte WebSphere MQ para SOAP despliegan un servicio web y lo ejecutan utilizando un escucha SOAP de WebSphere MQ . Las herramientas también generan WSDL, apéndices de cliente .NET y clases de proxy `.java` para desarrollar el transporte WebSphere MQ para clientes SOAP.

Siga estos pasos para crear un servicio y prepararlo para el despliegue y la generación de clientes. Siga los pasos de las tareas relacionadas para crear un servicio utilizando Eclipse o Microsoft Visual Studio 2008.

Procedimiento

1. Desarrolle el servicio utilizando el entorno de desarrollo normal.
2. Probar el servicio utilizando un cliente de servicios web HTTP
3. Siga estos pasos para preparar el directorio de despliegue:
   - Para Java

a. Copie el archivo `.java` que define la interfaz de servicio en el directorio de despliegue.

b. Copie los archivos `.class` del servicio en el directorio correspondiente al nombre del paquete.

c. Compruebe que la vía de acceso de clases puede localizar todas las clases necesarias: compile el archivo `.java` del servicio utilizando **javac**.

- Para .NET

a. Copie el archivo `.asmx` que define el servicio en el directorio de despliegue y

b. Si está utilizando el modelo de detrás de código, copie los archivos `.dll` en un directorio *deployment directory*`\bin` .

Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse

Desarrolle un servicio web Axis 1.4 para ejecutarlo utilizando WebSphere MQ como proveedor de servicios. Utilice el entorno de desarrollo de servicios web normal para crear un servicio para el despliegue en Axis 1.4.

Antes de empezar

Tenga en cuenta los requisitos para desplegar un servidor web en el escucha SOAP de WebSphere MQ para Axis 1.4.

- El escucha SOAP de WebSphere MQ para Axis 1.4 requiere un JRE de IBM de la versión 5.0 o superior. El JRE y el JDK utilizados para el desarrollo deben estar en el mismo nivel de versión.
- El escucha SOAP de WebSphere MQ para Axis 1.4 requiere el `axis.jar` instalado con WebSphere MQ. Cambie la vía de acceso de compilación en el entorno de desarrollo para que haga referencia al archivo `axis.jar` instalado con WebSphere MQ, en lugar de a los archivos `axis.jar` instalados con el entorno de desarrollo.
- Hasta, e incluyendo, WebSphere MQ V7.0.1, el WSDL generado para el servicio desplegado es RPC/codificado. A partir de V7.1, también puede solicitar WSDL de estilo RPC/literal. El WSDL generado sólo se utiliza para el despliegue. Puede definir el servicio utilizando WSDL compatible con WS-I.

Acerca de esta tarea

Cree el servicio utilizando el entorno de desarrollo de servicios web normal.

En esta tarea, utilizamos el Eclipse Java EE IDE for Web Developers de código abierto disponible libremente, conocido como Galileo. Para el servidor de aplicaciones, utilizamos WebSphere Application Server Community Edition v2.1 (Community Edition), basado en Geronimo. Consulte las tareas relacionadas para obtener información sobre cómo obtener, instalar y configurar el IDE y el servidor.

Pruebe el servicio utilizando HTTP como transporte utilizando el Explorador de servicios web proporcionado en el IDE antes. De forma alternativa, genere un proxy de cliente HTTP y pruebe el servicio utilizando su propio código de cliente.

Puede seguir estos pasos para desarrollar un servicio web ascendente. Utilice el programa de ejemplo `StockQuoteAxis.java` como ejemplo.

Procedimiento

1. Inicie Eclipse IDE para Java EE Developers con un nuevo espacio de trabajo.
2. Configure el espacio de trabajo para utilizar Java50,

   WebSphere Application Server Community Edition 2.1.4 no funciona con Java60.

   a) **Ventana** > **Preferencias** > **Java** > **JRE instalados** > **Añadir ...** > **VM estándar** > **Siguiente** > **Directorio ...**

   b) Vaya al directorio de instalación de **Java50** > **OK** > **Finalizar**

   c) Compruebe el JRE de **Java50** > Correcto

3. Añada el entorno de ejecución de Community Edition e inicie Community Edition.
   a) **Ventana** > **Preferencias** > **Servidor** > **Entornos de ejecución** > **Añadir ...**
   b) Seleccione **IBM WASCE v2.1** en la lista **Nuevo entorno de ejecución de servidor** > Comprobar **Crear un nuevo servidor local** > **Siguiente**

      Si **IBM WASCE 2.1** no está en la lista, debe completar otras dos tareas:

      i) Instale WebSphere Application Server Community Edition.

      ii) Instale la actualización de Eclipse para Community Edition.

      Encontrará los detalles en WebSphere Application Server Community Edition
   c) Vaya al directorio de instalación del servidor de aplicaciones > **Aceptar** > **Finalizar** > **Aceptar**.
   d) Pulse con el botón derecho del ratón en **IBM WASCE v2.1** en la vista de servidores > **Iniciar**

      Consejo: Puede gestionar WASCE en Eclipse: Pulse con el botón derecho del ratón en **IBM WASCE v2.1** > **Iniciar consola WASCE** . el valor predeterminado **`Username`** y **`Password`** son `system` y `manager` .
4. Configure el servidor y el tiempo de ejecución para los servicios web.
   a) **Ventana** > **Preferencias** > **Servicios Web** > **Servidor y tiempo de ejecución**
   b) Seleccione **IBM WASCE v2.1 Server** como servidor.
   c) Deje **Apache Axis** como tiempo de ejecución del servicio web.
5. Cree un proyecto web dinámico.
   a) **Archivo** > **Nuevo** > **Proyecto web dinámico**.

      Asigne el nombre `StockQuoteAxis`al proyecto.
   b) Seleccione **Añadir proyecto a un EAR** > **Nuevo ...**
   c) En la página **Proyecto de aplicación EAR** , escriba **`Project name`** `StockQuoteAxisEAR` > **Finalizar**

      Responda **Aceptar** en respuesta al recuadro de diálogo que le sugiere que cambie a la perspectiva Java EE o que permanezca en la perspectiva Java para seguir exactamente estas instrucciones.
   d) **IBM WASCE 2.1** está seleccionado como tiempo de ejecución de destino. Acépelo y los demás valores predeterminados > **Finalizar**.

      Responda **Aceptar** en respuesta al recuadro de diálogo que le sugiere que cambie a la perspectiva Java EE o que permanezca en la perspectiva Java para seguir exactamente estas instrucciones.
6. Importar el programa de ejemplo `StockQuoteAxis.java`
   a) Abra el proyecto web **StockQuoteAxis** > Pulse con el botón derecho del ratón en la carpeta **`src`** > **Importar ...**
   b) Seleccione **General** > **Sistema de archivos** > **Siguiente**
   c) Vaya a *`MQ_INSTALLATION_PATH`*`\tools\soap\samples\java\server` > check **`StockQuoteAxis.java`** > **Finalizar**

      `MQ_INSTALLATION_PATH` representa el directorio de alto nivel donde está instalado WebSphere MQ . Debe resaltar el directorio del servidor para ver los archivos que contiene.
7. Corrija el error de compilación moviendo `StockQuoteAxis.java` a su paquete correcto.
   a) Abra `StockQuoteAxis.java` y pulse con el botón derecho el problema > **Arreglo rápido**
   b) Efectúe una doble pulsación en **Move 'StockQuoteAxis.java' para empaquetar 'soap.server'** > **Guardar**.
8. Cree un servicio web desde `StockQuoteAxis.java` .
   a) Pulse con el botón derecho `StockQuoteAxis.java` > **Servicios web** > **Crear servicio web** > **Siguiente**.

      Acepte la configuración predeterminada para el servicio:

      Tipo de servicio web
      Servicio Java beanWeb ascendente

Implementación de servicio
soap.server.StockQuoteAxis

Servidor
`IBM WASCE v2.1 server`

Tiempo de ejecución de servicio web
Apache Axis

proyecto de servicio
Eje StockQuote

Proyecto EAR de servicio
StockQuoteAxisEAR

Configuración
Sin generación de cliente

9. Seleccione los métodos a los que acceder y el estilo del servicio web > **Siguiente**.

   Inicie el servidor si se le solicita.

   a) Deje todos los métodos seleccionados.

   b) Seleccione el estilo de documento/literal (envuelto).

10. **Finalizar**

    Cuando el servicio se haya desplegado, busque en la carpeta `WebContent\wsdl` del proyecto web de `StockQuoteAxis` y busque el archivo `StockQuoteAxis.wsdl` generado.

11. Pruebe el servicio utilizando HTTP con el Explorador de servicios web.

    a) Pulse con el botón derecho `StockQuoteAxis.wsdl` > **Probar con explorador de servicios web**.

    b) Pulse la operación **getQuote** en las acciones **StockQuoteAxisSoapEnlace** en la ventana **Explorador de servicios web** .

    c) Escriba `ibm` en el campo de entrada **symbol** > **Ir**

12. Pruebe el servicio utilizando el transporte WebSphere MQ para SOAP.

    Para desplegar el servicio, utilice **`SimpleJavaListener`**, que se encuentra en `com.ibm.mq.soap.jar`. Debe añadir las bibliotecas Java y SOAP de WebSphere MQ a la vía de acceso de construcción.

    a) Pulse con el botón derecho del ratón en el proyecto web **StockQuoteAxis** > **Vía de acceso de compilación** > **Configurar vía de acceso de compilación ...**

    b) Pulse el separador **Bibliotecas** > **Añadir Jars externos ...**. Vaya a *`MQ_INSTALLATION_PATH`*`\java\lib` . y seleccione todos los archivos `.jar` > **Abrir** > **Añadir Jars externos ...**. Vaya a *`WMQ Install directory`*`\java\lib\soap` y seleccione todos los archivos `.jar` > **Abrir** > **Aceptar**.

    `MQ_INSTALLATION_PATH` representa el directorio de alto nivel donde está instalado WebSphere MQ .

    c) En el Explorador de proyectos, pulse con el botón derecho `StockQuoteAxis\Java Resources\Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class\ SimpleJavaListener` > **Ejecutar como ...**> **Ejecutar configuraciones ...**

    Consejo:

    Si no hay ninguna configuración para `SimpleJavaListener` , pulse el icono **Nueva configuración** en la página **Crear, gestionar y ejecutar configuraciones** del asistente **Ejecutar configuraciones** .
    El **`SimpleJavaListener`** no tiene ningún mandato para detenerlo. Para supervisar o detener **`SimpleJavaListener`**, abra la **perspectiva Depurar** en Eclipse.

    d) Abra la pestaña **(x) = Argumentos** . En el área de entrada **Argumentos de programa** , escriba los parámetros en **`SimpleJavaListener`**.

    Para este ejemplo, escriba

```
-u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
```

Nota: El servicio de destino es `StockQuoteAxis` para que coincida con el nombre de servicio de destino creado en el descriptor de despliegue de servicio, `StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd` . amqwdeployWMQService crea un servicio de destino denominado `soap.server.StockQuoteAxis`. En este ejemplo, está utilizando los mismos `StockQuoteAxis.class` y `service-config.wsdd` que el servidor HTTP.

e) En la misma pestaña, configure **Directorio de trabajo** para que haga referencia al archivo `server-config.wsdd` : `${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}`

a) **Ejecutar**

Los errores se graban en la consola. Si la consola permanece en blanco, **`SimpleJavaListener`** se ha iniciado correctamente.

b) Para probar el despliegue, ejecute un cliente StockQuoteAxis, desarrollado en la tarea, “Desarrollo de un cliente JAX-RPC para el transporte WebSphere para SOAP utilizando Eclipse” en la página 993.

Ejemplo: programa de ejemplo de eje StockQuote

El servicio web Java de ejemplo, `StockQuoteAxis.java`, se instala en *`WMQ install directory`*`\tools\soap\samples\java\server` . `StockQuoteAxis.java`, Figura 170 en la página 984, tiene cuatro métodos:

1. `float getQuote(String symbol)`
2. `void getQuoteOneWay(String symbol).`
3. `int asyncQuote(int delay)`
4. `float getQuoteTran(String symbol)`

```
package soap.server;
import java.lang.Thread;
import java.io.FileWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }

    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,
                exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

Figura 170. Eje StockQuote

Qué hacer a continuación

Despliegue el servicio utilizando WebSphere MQ Transport for SOAP, en lugar de HTTP utilizando el mandato **amqwdeployWMQService**.

El mandato tiene una opción, `axisDeploy`, que despliega el servicio creando un descriptor de despliegue Apache Axis 1.4 . El escucha SOAP de WebSphere MQ ejecuta el servicio. El escucha SOAP se denomina escucha SimpleJavay se proporciona con el transporte WebSphere MQ para SOAP.

Tareas relacionadas

Desarrollo de un servicio .NET 1 o 2 para el transporte WebSphere MQ para SOAP utilizando Microsoft Visual Studio 2008
Desarrollar el servicio web SampleStockQuote para .NET 1 o .NET 2 utilizando Microsoft Visual Studio 2008

Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación de candidato W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones JEE. Esta tarea es el paso 2 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C .

Desarrollo de un servicio .NET 1 o 2 para el transporte WebSphere MQ para SOAP utilizando Microsoft Visual Studio 2008

Desarrollar el servicio web SampleStockQuote para .NET 1 o .NET 2 utilizando Microsoft Visual Studio 2008

Acerca de esta tarea

Cree el servicio StockQuote con una implementación de código posterior utilizando Visual Studio 2008.

Procedimiento

1. Cree una plantilla para el servicio y compruebe que se ejecuta en HTTP.
   a) Inicie Visual Studio 2008 > **Archivo** > **Nuevo** > **Proyecto ...**. Seleccione **C#** Tipo de proyecto, **, NET Framework 2**y **ASP.NET Aplicación de servicio web**. Escriba el **Nombre:** y **Nombre de solución:** `StockQuoteDotNet` > **Aceptar**
   b) Pulse con el botón derecho **Service1.asmx** en el **Explorador de soluciones** > **Renombrar** > `StockQuote.asmx`.
   c) Cambie el fragmento de código `public class Service1` por `public class StockQuote`.
   d) Pulse con el botón derecho **StockQuote.asmx** en el **Explorador de soluciones** > **Abrir con ...** > **Editor XML**. Cambie `Class="StockQuoteDotNet.Service1"` por `Class="StockQuoteDotNet.StockQuote"`
   e) Cambie el fragmento de código `[WebService(Namespace = "http://tempuri.org/")]` por `[WebService(Namespace = "http://stock.samples/")]`.
   f) Elimine la línea de código `[ToolboxItem(false)]`.
   g) Compruebe que todo sea correcto hasta ahora: **Debug** > **Start Debugging (F5)**. Verifique la salida en el Explorador.
2. Añada los métodos del ejemplo `SQDNNonInline.asmx.cs`y pruebe el servicio en HTTP.
   a) Abra *MQ_INSTALLATION_PATH*`\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs` y sustituya el método `HelloWorld` por los cuatro métodos `Quote` ; consulte Figura 171 en la página 986. *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ .
   b) **Compilación** > **Reconstruir** la solución > Pulse con el botón derecho del ratón en una de las **Hebra** con error > **Resolver** > Utilización de **System.Threading**.
   c) Pulse F5 para iniciar la depuración.

      El servicio no es compatible con WS-I Basic Profile v1.1. Tiene la opción de cambiar la anotación WebMethod de `[SoapRpcMethod]` a `[SoapDocumentMethod]`o de eliminar la anotación `[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]`.
   d) Pulse F5 para verificar la implementación utilizando HTTP.
3. Genere WSDL, clientes y ejecute el servicio utilizando el transporte WebSphere MQ para SOAP.
   a) Abra una ventana de mandatos en el árbol de directorios del proyecto, donde se almacena `StockQuote.asmx` .
   b) (Opcional) Utilice amqwdeployWMQService para generar artefactos. El gestor de colas debe estar iniciado:

      ```
      amqswdeployWMQService -f StockQuote.asmx
      -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
      &connectionFactory=()
      &destination=REQUESTDOTNET@QM1
      &targetService=StockQuote.asmx"
      StockQuote.asmx StockQuote.wsdl
      ```

      Todos los artefactos se crean en el árbol de directorios `./generated` .
   c) (Opcional) Genere sólo el WSDL para llamar al servicio utilizando el transporte WebSphere MQ para SOAP.

      ```
      amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
      &connectionFactory=()
      &destination=REQUESTDOTNET@QM1
      &targetService=StockQuote.asmx"
      StockQuote.asmx StockQuote.wsdl
      ```
   a) Ejecute el escucha .NET. Utilice `.\generated\server\startWMQNListener.cmd` o escriba el mandato:

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. Pruebe el servicio utilizando un cliente generado a partir del WSDL, o utilizando los clientes generados por **amqwdeployWMQService**.

Código de ejemplo

El servicio web .NET de ejemplo, StockQuoteDotNet, se instala en *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet. *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ . El enlace de servicio web de los ejemplos publicados difiere ligeramente del enlace utilizado en la tarea. La tarea utiliza los valores predeterminados utilizados en Visual Studio 2008.

Hay dos ejemplos de servicios web .NET Framework 1 y .NET Framework 2. StockQuoteDotNet.asmx es un servicio en línea. SQDNNoninline.asmx es un servicio web detrás de código implementado por SQDNNoninline.asmx.cs.

StockQuoteDotNet tiene cuatro métodos:

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteDOC(String symbol)

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

Figura 171. Servicio en línea: StockQuoteDotNet.asmx

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Figura 172. Detrás de código: Diseño SQDNNonInline.asmx

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

Figura 173. Detrás de código: Implementación: SQDNNonInline.asmx.cs

Tareas relacionadas

Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse
Desarrolle un servicio web Axis 1.4 para ejecutarlo utilizando WebSphere MQ como proveedor de servicios. Utilice el entorno de desarrollo de servicios web normal para crear un servicio para el despliegue en Axis 1.4.

Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación de candidato W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones JEE. Esta tarea es el paso 2 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C .

Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS

Un servicio web enlazado a la recomendación de candidato W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones JEE. Esta tarea es el paso 2 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C .

Antes de empezar

Utilice Rational Application Developer para crear el servicio web EJB. El asistente de servicios web en Rational Application Developer tiene una opción para crear un servicio web utilizando la recomendación de candidato W3C para el enlace SOAP sobre JMS. Se necesita Rational Application Developer 7.54 .
El ejercicio ha utilizado Rational Application Developer incluido en Rational Software Architect para WebSphere Software v7.5.5.1,

El EJB se despliega en WebSphere Application Server desde Rational Application Developer como parte de esta tarea. Debe completar “Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS” en la página 1025

Para crear el WSDL utilizado realmente en la tarea, primero debe completar la tarea, “Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse” en la página 980. A continuación, puede importar el WSDL desde el proyecto web dinámico en el espacio de trabajo Galileo de Eclipse o desde el servicio web HTTP en ejecución desplegado en WASCE.

Es posible que WebSphere Application Server siga ejecutándose como resultado de realizar “Configurar recursos de WebSphere Application Server” en la página 1027. Si no lo está, puede iniciarlo desde la vista Servidores en RAD.

Acerca de esta tarea

En esta tarea, puede volver a desplegar el servicio `StockQuoteAxis` para que no se ejecute como un servicio Axis JAX-RPC ejecutado por **`SimpleJavaListener`** utilizando el transporte WebSphere MQ para SOAP, para que sea un servicio JAX-WS que se ejecuta en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C .

Hay dos partes para migrar el servicio desde **`SimpleJavaListener`** a WebSphere Application Server:

1. Genere la interfaz de servicio web a partir del WSDL para el servicio utilizando el asistente de servicio web EJB descendente en Rational Application Developer.
2. Implementación del servicio importando el ejemplo SOAP de WebSphere MQ `StockQuoteAxis.java`.

Un enfoque alternativo habría sido generar el servicio de abajo a arriba, desde `StockQuoteAxis.java`. Sin embargo, para asegurarse de que la interfaz para el servicio migrado es idéntica, el enfoque descendente es mejor, ya que utiliza el mismo WSDL.

El servicio web se desarrolla para el contenedor EJB y no para el contenedor web porque el soporte JMS forma parte del contenedor EJB.

Procedimiento

1. Inicie Rational Application Developery verifique que WebSphere Application Server se está ejecutando.
   a) Inicie Rational Application Developer en un nuevo espacio de trabajo.
   b) Abra la perspectiva Java EE .
   c) Abra la pestaña **Servidores** y seleccione WebSphere Application Server está en ejecución.
      - Si no hay ningún WebSphere Application Server v7.0 en la vista, pulse con el botón derecho del ratón en la vista > **Nuevo** > **Servidor**. Siga las opciones del asistente para crear una instancia de WebSphere Application Server v7.0 .
      - Si el servidor está presente, pero no se ha iniciado, pulse la punta de flecha para iniciarlo.
      - Para verificar las propiedades y obtener acceso rápido a los registros del servidor, pulse con el botón derecho **WebSphere Application Server v7.0 en localhost** > **Propiedades** > **WebSphere Application Server**.
      - Para administrar el servidor, utilice un navegador externo y abra el URL, `http://localhost:9061/ibm/console/unsecureLogon.jsp`, o pulse con el botón derecho del ratón en **WebSphere Application Server v7.0 en localhost** > **Ejecutar consola administrativa**.
      - El valor predeterminado es publicar automáticamente. Muchas personas prefieren desplegar actualizaciones en el servidor manualmente. Efectúe una doble pulsación en **WebSphere Application Server v7.0 en localhost**y expanda **Publicación** twisty en la ventana**Visión general** . Pulse **Nunca publicar automáticamente**.
      - Otro valor predeterminado que es posible que desee modificar es desmarcar el recuadro de selección **Terminar servidor al concluir el entorno de trabajo** en la ventana **Visión general** .

2. Crear los proyectos JEE

   Debe crear un EAR (Enterprise Application Project) y un EJB (Enterprise Java Bean).

   a) **Archivo** > **Nuevo** > **Enterprise Application Project**. Asigne el nombre `W3CJMSEAR` > **Finalizar**al proyecto.

      Los valores predeterminados deben identificar WebSphere Application Server v7.0 como el tiempo de ejecución de destino y la versión EAR 5.0. Se debe seleccionar la configuración predeterminada.

   b) **Archivo** > **Nuevo** > **Proyecto EJB**. Asigne el nombre `W3CJMSEJB`al proyecto. Seleccione `W3CEARJMS` como **Nombre de proyecto EAR** > **Siguiente**.

      La versión predeterminada del módulo EJB es 3.0 y se vuelve a utilizar la configuración predeterminada.

   c) Desmarque el recuadro de selección **Crear un módulo JAR de cliente EJB** > **Finalizar**.

3. Genere y despliegue el servicio web EJB a partir del WSDL `StockQuoteAxis` .

   a) **Ejecutar** > **Iniciar el explorador de servicios web**.

   b) Seleccione la página WSDL utilizando los iconos de la ventana**Explorador de servicios web** > pulse **WSDL principal** en el Navigator.

   c) En la ventana **Acciones** , escriba o vaya al URL de WSDL para `StockQuoteAxis.wsdl`.

      Si tiene WASCE en ejecución con `StockQuoteAxis` desplegado como un servicio HTTP, el URL es:

      ```
      http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
      ```

      Si tiene el WSDL en el sistema de archivos, el URL puede ser:

      ```
      File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
      ```

   d) Pulse la línea que contiene el URL importado en el árbol de Navigator .

      Es la línea que sigue inmediatamente a **WSDL principal**, si es el primer WSDL que ha importado en el Explorador de servicios web.

   e) En la ventana **Acciones** , pulse **Iniciar asistente de servicio web** > **Esqueleto de servicio web** > **Ir**

   f) En el asistente Servicio web, seleccione **Servicio web EJB descendente**

      Seleccione o verifique la configuración utilizando la información de Tabla 141 en la página 989 Comprobar **Sobrescribir archivos sin avisar** > **Siguiente**.

Tabla 141. Configuración de servicio web EJB descendente	
Campo	**Valor**
Servidor	`WebSphere Application Server v7.0`
Tiempo de ejecución de servicio web	`IBM WebSphere JAX-WS`
proyecto de servicio	`W3CJMSEJB`
Proyecto EAR de servicio	`W3CJMSEAR`
Configuración:	`No client generation`

   g) En la página subtitulada, **Especificar opciones para crear un servicio web WebSphere JAX-WS EJB Top Down**, marque el recuadro **Cambiar a enlace JMS**. Seleccione también **Habilitar estilo de derivador**, **Copiar WSDL en proyecto** y **Generar descriptor de despliegue de servicio web** > **Siguiente**.

   h) En la página titulada, **WebSphere**, seleccione **Utilizar protocolo de interoperatividad SOAP/JMS** y proporcione valores de Tabla 142 en la página 990, dejando otros campos en blanco > **Siguiente**.

Tabla 142. WebSphere Configuración de enlace JMS JAX-WS	
Campo	**Valor**
Destino de JMS	queue
Nombre JNDI del destino:	requestaxis
Fábrica de conexiones JMS	qm1
Responder a nombre	W3CJMSEAR
Configuración:	replyaxis

a) En la página titulada, **WebSphere**, escriba qm1as en el campo **ActivationSpec Nombre JNDI** > **Siguiente**.

RAD tarda unos 30 segundos a un minuto en generar y desplegar el proyecto.

b) Ignore las opciones de la página **Publicación de servicio web** > **Finalizar**.

4. Compruebe el WSDL generado.

Ha solicitado que se genere y guarde el WSDL específico del servicio en el proyecto.

a) En el navegador Enterprise Explorer, abra la carpeta **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl**. Efectúe una doble pulsación en StockQuoteAxis.wsdl para abrirlo en el editor WSDL.

Inspeccione el enlace; verá el URL de JMS:

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. Paso opcional: enlace el EJB a SOAP a través de HTTP utilizando JAX-WS.

Proporcionar dos enlaces al EJB ofrece a los clientes la opción de enlaces SOAP para llamar al servicio web. También proporciona a los clientes los medios para consultar el servidor web para obtener su WSDL, utilizando HTTP.

Los pasos para enlazar un EJB a SOAP a través de HTTP no se incluyen como parte de la tarea.

6. Implementar y volver a desplegar StockQuoteAxis utilizando el ejemplo StockQuoteAxis.java

a) En el navegador Enterprise Explorer, abra la carpeta **W3CJMSEJB** > **Services** Efectúe una doble pulsación en StockQuoteAxisService para abrir la clase de implementación en un editor Java.

b) Abra el programa de ejemplo StockQuoteAxis.java en la carpeta *WebSphere MQ Installation directory*\tools\soap\samples\java\server > Seleccionar todos los métodos, pero no el nombre de clase > **Copiar**.

c) En StockQuoteAxisSoapBindingImpl.java , seleccione todos los métodos, pero no el nombre de clase, y pegue los métodos de StockQuoteAxis.java.

d) Añada una sentencia de impresión a la salida de la consola de WebSphere Application Server cuando se llame al servicio.
Cambie el método getQuote(símbolo de serie):

```
public float getQuote(String symbol) {
    System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
                       + symbol);
    return ((float) 55.25);
```

e) Corrija las importaciones: **Origen** > **Organizar importaciones** > **Guardar**.

f) Corrija los tres errores debido a que la implementación no coincide con la interfaz.

Los errores se deben a tres de los métodos de StockQuoteAxis.java que generan excepciones, y al WSDL para el servicio que no contiene ningún mensaje de error. El problema se diagnostica como una discrepancia entre las firmas de método y las anotaciones de servicio web de método.

Anote los métodos con @WebFault y vuelva a generar el WSDL, o mantenga la interfaz sin modificar y elimine las excepciones.

Para mantener la interfaz igual, elimine los tres `throws exception` de las firmas de método > **Guardar**.

Qué hacer a continuación

"Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS" en la página 1036

Tareas relacionadas

Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse
Desarrolle un servicio web Axis 1.4 para ejecutarlo utilizando WebSphere MQ como proveedor de servicios. Utilice el entorno de desarrollo de servicios web normal para crear un servicio para el despliegue en Axis 1.4.

Desarrollo de un servicio .NET 1 o 2 para el transporte WebSphere MQ para SOAP utilizando Microsoft Visual Studio 2008
Desarrollar el servicio web SampleStockQuote para .NET 1 o .NET 2 utilizando Microsoft Visual Studio 2008

Desarrollo de clientes de servicio web de WebSphere MQ para el transporte de WebSphere MQ para SOAP

Utilice el entorno de desarrollo normal para desarrollar clientes de servicios web para utilizarlos con el transporte WebSphere MQ para SOAP.

Antes de empezar

Cree el servicio. Puede utilizar uno de los ejemplos de "Desarrollo de servicios web para el transporte de WebSphere MQ para SOAP" en la página 979.

Elija cómo va a desarrollar, desplegar y utilizar los clientes y dónde ir para obtener el WSDL para la generación de clientes.

Decida su enfoque para desarrollar clientes y servicios para el transporte de WebSphere MQ para SOAP.

Hay dos enfoques.

1. Utilice las herramientas de desarrollo estándar, desarrolle un servicio y cliente HTTP y, a continuación, utilice el URL para el transporte WebSphere MQ para SOAP.
2. Utilice las herramientas y los ejemplos proporcionados con el transporte WebSphere MQ para SOAP.

Si toma la ruta HTTP, puede ejecutar el servicio en un servidor HTTP y también ejecutarlo utilizando el transporte WebSphere MQ para SOAP. Para ejecutarlo utilizando el transporte WebSphere MQ para SOAP, configure el escucha de WebSphere MQ adecuado para SOAP y configure las vías de acceso y los descriptores de despliegue para ejecutar el servicio. Las herramientas proporcionadas por WebSphere MQ Transport para SOAP realizan la configuración. De forma alternativa, puede configurar el entorno para ejecutar los escuchas.

Las herramientas proporcionadas con WebSphere MQ Transport para SOAP son útiles para empezar y aprender a desplegar el transporte. Para el trabajo de producción, existen ventajas en el uso de herramientas estándar y el despliegue del mismo servicio accesible para distintos transportes SOAP.

Decidir el tipo de cliente a desarrollar

Debe decidir qué tipo de cliente de servicio web desarrollar. La elección depende de si conoce la interfaz de servicio y la dirección del servicio.

Si se conoce la interfaz, utilice las herramientas Axis o .NET para generar clases de cliente proxy desde la interfaz de servicio. Las clases de cliente proxy facilitan la escritura de un cliente para llamar al servicio. Si la ubicación del servicio se conoce al desarrollar el cliente, utilice la interfaz de proxy estático. Si la ubicación del servicio cambia, por ejemplo, si el servicio se vuelve a desplegar en un servidor de producción, utilice la interfaz de proxy dinámico.

Si la interfaz de servicio no se conoce en el momento en que desarrolla un cliente, en Axis, puede crear un cliente DII (Dynamic Invocation Interface) para Axis 1.4. Un cliente DII utiliza una interfaz genérica para llamar a cualquier servicio. Para pasar parámetros a un servicio determinado correctamente, debe crear la interfaz de servicio específica mediante programación. Cree la interfaz mediante programación en el cliente, o cargando el WSDL para el servicio en el cliente. En Axis2, puede crear un cliente Dispatch. El cliente Dispatch utiliza un modelo de documento para describir la solicitud del cliente, mientras que un cliente DII utiliza un modelo de llamada. Ambos trabajan en la creación dinámica de la solicitud.

Obtener el WSDL para el servicio

Excepto en el caso de la interfaz de servicio que se está creando mediante programación, primero debe obtener el WSDL de servicio para crear un cliente de servicio web. El WSDL de servicio se puede obtener a partir de tres orígenes diferentes:

1. Directamente desde la implementación del servicio web utilizando una herramienta como **java2wsdl** (Axis) o **disco** (.NET).
2. Consultando el servicio web utilizando el URL: *Web service http url*?wsdl.
3. Desde un archivo, ya sea en un sistema de archivos, o desde un registro como UDDI o WebSphere Service Registry and Repository.

Nota: Si no se puede acceder al servicio utilizando HTTP, la consulta WSDL no funciona. Es posible que el propio servicio solo esté disponible utilizando el transporte WebSphere MQ para SOAP.

El WSDL generado por **amqwdeployWMQService** no es el mismo que el WSDL generado utilizando **java2wsdl** o **disco**. El WSDL generado también es diferente de cualquier WSDL con el que se haya iniciado para crear el servicio "Arriba". En Axis, el descriptor de despliegue `server-config.wsdd` correlaciona el mensaje SOAP generado por un cliente con una operación y un servicio. **amqwdeployWMQService** genera un descriptor de despliegue diferente de Eclipse.

El WSDL que utilice para crear clientes depende de cómo se despliegue el servicio:

desplegado con amqwdeployWMQService
Utilice el WSDL generado por **amqwdeployWMQService** . Especifique el distintivo -w y seleccione `rpcLiteral` WSDL. Para la compatibilidad, puede seleccionar `rpcEncoded` WSDL. `rpcEncoded` WSDL sólo funciona con clientes .NET y Axis 1.4 .

Despliegue manual utilizando SimpleJavaListener
Utilice uno de los siguientes archivos WSDL:

1. WSDL utilizado para definir el servicio o almacenado en un repositorio.
2. WSDL generado a partir del servicio por **java2wsdl** .
3. WSDL consultado utilizando el URL *Web service http url* ?wsdl, si está disponible desde un servidor HTTP. Puede ejecutar una herramienta como, por ejemplo, el Explorador de servicios web para importar la definición de servicio directamente en Eclipse.

Es posible que tenga que cambiar el URI del servicio. Cámbielo de la dirección del servicio HTTP al URI para el transporte WebSphere MQ para SOAP.

Despliegue manual utilizando amqSOAPNETListener.
Utilice uno de los siguientes archivos WSDL:

1. WSDL utilizado para definir el servicio o almacenado en un repositorio.
2. WSDL obtenido de la clase de servicio .NET (.asmx). utilizando **disco**.
3. WSDL consultado utilizando el URL *Web service http url* ?wsdl, si está disponible. Puede ejecutar una herramienta como, por ejemplo, el Explorador de servicios web para importar la definición de servicio directamente en Eclipse.
4. WSDL obtenido ejecutando **amqswsdl** en la clase de servicio .NET (.asmx).

Es posible que tenga que cambiar el URI del servicio. Cámbielo de la dirección del servicio HTTP al URI para el transporte WebSphere MQ para SOAP.

Desplegado en Windows Communication Foundation
Obtenga el WSDL de servicio utilizando el URL *Web service http url*?wsdl. El servicio debe definirse con la configuración del comportamiento de serviceMetaDatos como parte de la definición de servicio.

Despliegue en una plataforma de servidor diferente.
Siga las instrucciones proporcionadas con la plataforma sobre cómo obtener el WSDL de servicio correcto.

Acerca de esta tarea

Desarrollar clientes utilizando herramientas de desarrollo estándar. Las tareas siguientes ilustran cómo crear clientes para .NET 1 y 2, Axis 1.4 (JAX-RPC) y Axis2 (JAX-WS). Para Windows Communication Foundation, consulte los enlaces de tareas relacionadas.

Desarrollo de un cliente JAX-RPC para el transporte WebSphere para SOAP utilizando Eclipse

Desarrolle un cliente de servicio web Axis 1.4 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Antes de empezar

Debe tener el servicio disponible. Si está siguiendo la tarea como un ejercicio práctico, utilice el espacio de trabajo y el servicio que ha creado en la tarea, "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980. Debe tener un servidor de aplicaciones en ejecución en Eclipse que dé soporte a los servicios web de Axis 1.4 . En esta tarea utilizamos el WebSphere Application Server Community Edition Versión 2.1.4disponible de forma gratuita. Se configura como parte de la tarea "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980. También puede utilizar Tomcat 6, que es un servidor de aplicaciones de código abierto más pequeño.

Acerca de esta tarea

La tarea muestra el desarrollo de tres tipos de cliente para el servicio Axis StockQuotede ejemplo utilizando Eclipse que se ejecuta en Windows. Los clientes son un cliente estático y dinámico desarrollado utilizando el proxy de cliente y un cliente DII.

Se ilustran dos enfoques alternativos para generar los proxies de cliente a partir de WSDL:

1. Generación de proxies de cliente utilizando **amqwdeployWMQService** .
2. Importación de WSDL en Eclipsey utilización del asistente de servicios web para generar los proxies de cliente.

Procedimiento

1. Inicie el Eclipse IDE para desarrolladores de Java EE .
2. Cree un proyecto Java denominado StockQuoteAxisClient:
   a) Cambie a la perspectiva Java > **Archivo** > **Nuevo** > **Proyecto Java**. En el campo **Project name** de la página **Crear un proyecto Java** , escriba StockQuoteAxisEclipseClient. Asegúrese de que el entorno de ejecución sea **J2SE1-1.4** o **J2SE-1.5** > **Siguiente** .
   b) En la página **Valores de Java** , seleccione el separador **Bibliotecas** > **Añadir JAR externos ...**
   c) Vaya a *MQ_INSTALLATION_PATH*/java/lib y seleccione todos los archivos .jar > **Abrir**.
   MQ_INSTALLATION_PATH es el directorio donde está instalado WebSphere MQ.
   d) Vaya a *MQ_INSTALLATION_PATH*/java/lib/soap y seleccione todos los archivos .jar > **Abrir**. Debe haber instalado axis.jar desde el soporte de instalación de WebSphere MQ en este directorio.
   MQ_INSTALLATION_PATH es el directorio donde está instalado WebSphere MQ.

e) La pestaña **Biblioteca** ahora hace referencia a todos los archivos `.jar` necesarios para crear el cliente > **Finalizar**.

3. Siga uno de estos dos métodos para crear proxies en Eclipse para el servicio web del eje StockQuotede ejemplo:

   - Genere los proxies de cliente utilizando **amqwdeployWMQService** .

     a. Cree un gestor de colas. Para la tarea, cree `QM1` como gestor de colas predeterminado.

     b. Cree un directorio de trabajo, `samples`. Copie el programa de ejemplo `StockQuoteAxis.java` en `samples/soap/server`.

     c. Modifique `amqwsetcp.cmd` en *MQ_INSTALLATION_PATH*`/bin` para incluir el directorio actual en la vía de acceso de clases. *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ .

     d. Abra una ventana de mandatos en `samples` y ejecute el mandato **amqwsetcp** modificado

     e. Cree WSDL para el servicio de eje StockQuoteejecutando el mandato,

     ```
     amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
     -u "jms:/queue?destination=REQUESTAXIS
     &initialContextFactory=com.ibm.mq.jms.Nojndi
     &connectionFactory=(connectQueueManager(QM1)binding(auto))"
     ```

     Recuerde: Utilice "`/`", en lugar de "`.`" o "`\`" cuando utilice mandatos Java.

     Consejo: En lugar de importar los proxies generados en Eclipse, puede importar el WSDL generado desde `.samples/generated`. Los proxies resultantes difieren de dos maneras:

     i) Los nombres de los paquetes son diferentes, lo que puede refactorizar.

     ii) Los proxies generados de Eclipse incluyen una clase de ayudante adicional, `StockQuoteAxisProxy.java`

     f. Cree los proxies de cliente para el servicio Axis StockQuoteejecutando el mandato:

     ```
     amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
     -u "jms:/queue?destination=REQUESTAXIS
     &initialContextFactory=com.ibm.mq.jms.Nojndi
     &connectionFactory=(connectQueueManager(QM1)binding(auto))"
     ```

     g. Importe los proxies de cliente en StockQuoteAxisClient:

     i) Pulse con el botón derecho **StockQuoteAxisClient\src** > Seleccionar **Sistema de archivos** > **Siguiente** > **Examinar ...** > buscar la carpeta `.\samples\generated\client\remote\soap\server` > **Aceptar**.

     ii) Seleccione **servidor** en la página**Importar** > **Finalizar**.

     h. Refactorizar el nombre de paquete a `soap.server`.

     i) Pulse con el botón derecho del ratón en el paquete que contiene los proxies de cliente > **Refactorizar** > **Renombrar**. Escriba **New name**: `soap.server` > deje los valores predeterminados seleccionados para las otras opciones > **Aceptar** . Se han arreglado todos los errores.

   - Genere los proxies de cliente utilizando Eclipse.

     Puede elegir entre varias formas de obtener el WSDL para el servicio. En este ejemplo, el servicio se ha desplegado en WebSphere Application Server Community Edition y obtiene el WSDL del servidor web. El despliegue se describe en la tarea "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980,

     a. En Eclipse, cambie a la perspectiva Web y compruebe que WebSphere Application Server Community Edition v2.1 Server se esté ejecutando y que StockQuoteAxis esté desplegado y sincronizado.

     b. Importe el WSDL en el Explorador de servicios web:

i) Pulse el icono **Explorador de servicios web** en la barra de acciones o pulse **Ejecutar** > **Iniciar el Explorador de servicios web**.

ii) Pulse el icono de página WSDL en el Explorador de servicios web para conmutar a la página WSDL.

iii) Pulse **WSDL principal** en la ventana Navigator del Explorador de servicios web.

iv) Escriba el URL del servicio web, seguido de ?WSDL . El URL del eje StockQuote, desplegado en la tarea "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980, es:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

c. Genere los proxies de cliente:

i) En el navegador Explorador de servicios web, pulse **http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl** .

ii) En la ventana **Acciones** , pulse **Iniciar asistente de servicio web** > dejar **Cliente de servicio web** seleccionado > **Ir**.

iii) En la primera página del asistente, pulse el enlace de proyecto **Cliente** en la configuración > Seleccionar el proyecto de cliente **StockQuoteAxisClient** > **Aceptar**.

Consejo: Es posible que la ventana del asistente pierda el foco. Es necesario volver a poner el foco manualmente.

iv) El tiempo de ejecución del servicio web debe ser Apache Axis para generar un cliente JAX-RPC.

v) Pulse **Finalizar**.

vi) Cambie el URL estático del servicio para que apunte al transporte de WebSphere MQ para la dirección SOAP para el servicio de eje StockQuote. Puede optar por omitir este paso, hasta que haya probado el cliente con un servidor HTTP.

a) Abra `StockQuoteAxisServiceLocator.java` y busque la declaración para `StockQuoteAxis_address`.

b) Cambie el URL por

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

Consejo: Eclipse transforma automáticamente & en `&`, y al revés, al copiar y pegar series en código `.java` .

d. Cree tres clases de cliente Java, cada una con un método principal:

i) Crear paquete. Pulse con el botón derecho **StockQuoteAxisClient/src** > **Paquete nuevo**. Denomine `soap.client` > **Finalizar**.

ii) Seleccione **soap.client** > **New** > **Class**. Asigne un nombre a la clase `SQAStaticClient` > Check **public static void main (string [] args)** > Finish

iii) Repita el procedimiento para crear `SQADynamicClient.java` y `SQADIIClient.java`

e. Escriba el código de cliente.

Figura 177 en la página 999 a Figura 181 en la página 1001 proporcionan ejemplos de los tres estilos de código de cliente. Los ejemplos utilizan un URL HTTP para probar el cliente utilizando el servicio Axis StockQuotedesplegado en un servidor HTTP. Para ejecutar los clientes en el servicio Axis StockQuotedesplegado utilizando el transporte WebSphere MQ para SOAP, cambie el URL por:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
```

```
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

- – Figura 177 en la página 999 y Figura 179 en la página 1000 utilizan el proxy generado por Eclipse, que tiene la clase auxiliar StockQuoteadicional de Axisproxy que facilita un poco la codificación.
- – Figura 178 en la página 1000 y Figura 180 en la página 1000 utilizan el proxy generado por **amqwdeployWMQService** .
- – Figura 181 en la página 1001 no utiliza clases de proxy.

Cada uno de los clientes llama a `com.ibm.mq.soap.Register.extension()` para enlazar con el transporte de WebSphere MQ para SOAP. La extensión se registra en el descriptor de despliegue del cliente. El despliegue de cliente en Axis 1.4 se describe en "Despliegue de un cliente de servicio web en Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP" en la página 1031.

f. Ejecute los clientes enviando la solicitud SOAP a StockQuoteAxis alojado en el servidor de WebSphere Application Server Community Edition configurado en el espacio de trabajo.

i) Compruebe que el servidor se está ejecutando, StockQuoteAxis está desplegado y sincronizado.

ii) Seleccione o abra el cliente que desea probar > Pulse **Ejecutar** en la barra de acciones. De forma alternativa, pulse el icono de ejecución verde u ocho pulse el cliente en el navegador > **Ejecutar como** > **Ejecutar configuraciones ...**. Configure los parámetros que necesita para ejecutar el cliente.

g. Ejecute el cliente utilizando el transporte WebSphere MQ para SOAP.

El procedimiento utiliza **amqwdeployWMQService** para desplegar el servicio y sólo funciona con el cliente que utiliza el WSDL o proxies creados por **amqwdeployWMQService** . Para ejecutar el cliente utilizando el WSDL original, o proxies creados por eclipse, despliegue el servicio con su descriptor de despliegue creado por Eclipse. Inicie manualmente **SimpleJavaListener** utilizando el nombre de enlace de puerto de servicio como `targetServiceName` .

i) Siga las instrucciones de "Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService" en la página 1020 para desplegar el servicio en el escucha SOAP Java simple de WebSphere MQ . El despliegue de servicio sólo funciona para el cliente utilizando el WSDL o los proxies de cliente creados por **amqwdeployWMQService**.

ii) En una ventana de mandatos, ejecute **amqwclientconfig** para crear el archivo de descriptor de despliegue de cliente, `client-deploy.wsdd` .

iii) Importe `client-deploy.wsdd` en la raíz del proyecto Java que desea probar utilizando el transporte WebSphere MQ para SOAP.

a) Pulse con el botón derecho del ratón en el proyecto Java **StockQuoteAxisEclipseClient** > **Importar** > **Sistema de archivos** > **Siguiente** > **Examinar ...**

b) Vaya al directorio que contiene `client-deploy.wsdd` > **Abrir** > Seleccione el directorio en la página del asistente **Importar** > marque `client-deploy.wsdd` en el panel derecho.

c) Verifique que **En carpeta:** ha `StockQuoteAxisEclipseClient` entrado > **Finalizar**.

iv) Confirme que el directorio de trabajo para ejecutar una aplicación Java en este proyecto es el directorio `StockQuoteAxisEclipseClient` :

Pulse con el botón derecho del ratón en el proyecto Java **StockQuoteAxisEclipseCliente** > **Ejecutar como ....** > **Configuraciones de ejecución ...** > Seleccione el separador **(x) = Argumentos** > Verifique que en Directorio de trabajo el botón de selección **Predeterminado** esté seleccionado y la vía de acceso sea `StockQuoteAxisEclipseClient` . De forma alternativa, elija una de las opciones siguientes para seleccionar una ubicación o archivo diferente que contenga la configuración de cliente:

- Seleccione **Otros:** > escriba una vía de acceso de directorio de su elección.
- En la ventana **Argumentos de VM** , escriba `-Daxis.ClientConfigFile=`*`full path to client deployment descriptor file`*

v) Asegúrese de que el URL esté configurado para apuntar al servicio desplegado utilizando el transporte WebSphere MQ para SOAP. Ejecute el cliente tal como se describe en el paso ii.

Consejo: Normalmente, puede encontrar uno de estos errores:

i) `Exception: No client transport named 'jms' found!` .

ii) Error de conexión JMS.

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

Explicaciones:

i) `client-config.wsdd` no se encuentra o incluye la línea `<transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/>` en `client-config.wsdd`.

ii) Posiblemente un problema de vía de acceso de construcción-sin incluir los archivos .jar en *MQ_INSTALLATION_PATH*`/java/lib` . *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ.

iii) Problema de despliegue de servicio, ya sea con `server-config-wsdd` , o con parámetros pasados a **SimpleSoapListener** .

iv) Discrepancia entre el descriptor de despliegue y la implementación del servicio.

Si tiene dificultades para ejecutar el cliente en Eclipse, intente utilizar una ventana de mandatos:

i) Cambie al directorio `StockQuoteAxisEclipseClient\bin` en el árbol de directorios del espacio de trabajo.

ii) Ejecute **amqwsetcp** y **amqwclientconfig**

iii) Ejecute `java soap/client/SQAStaticClient`.

Clientes de servicio web JAX-RPC de ejemplo

Los clientes de servicio web Java de ejemplo que se suministran con WebSphere MQ se instalan en *MQ_INSTALLATION_PATH* `\tools\soap\samples\java\clients`. *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ .

SQAxis2Axis.java
`SQAxis2Axis.java`, Figura 174 en la página 998 , es un cliente proxy dinámico para invocar el servicio `StockQuoteAxis` . Puede alterar temporalmente el URL del servicio, que se compila en el proxy dinámico, proporcionando un URL en la línea de mandatos.

SQAxis2DotNet.java
`SQAxis2DotNet.java`, Figura 175 en la página 998, es un cliente proxy dinámico para invocar el servicio `StockQuoteDotNet` . Puede alterar temporalmente el URL del servicio, que se compila en el proxy dinámico, proporcionando un URL en la línea de mandatos.

WsdlClient.java
`WsdlClient.java`, Figura 176 en la página 999, es un cliente de invocación dinámica para invocar el servicio `StockQuoteDotNet` o `StockQuoteAxis` . El cliente invoca el servicio `StockQuoteAxis` de forma predeterminada. Añada la opción de línea de mandatos -D para invocar el servicio `StockQuoteDotNet` y -w para proporcionar un puerto diferente al de `.\generated\StockQuoteDotNet_Wmq.wsdl`

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
```

Figura 174. SQAxis2Axis.java

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figura 175. SQAxis2DotNet.java

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("  '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figura 176. WsdlClient.java

Los clientes de ejemplo utilizados en esta tarea:

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 177. Cliente estático utilizando el proxy generado de Eclipse

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 178. Cliente estático utilizando el proxy generado amqwdeployWMQService

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 179. Cliente dinámico utilizando el proxy generado de Eclipse

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 180. Cliente dinámico utilizando el proxy generado amqwdeployWMQService

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 181. Cliente DII (sin proxy)

Tareas relacionadas

Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse
Desarrollar un cliente de servicio web Axis2 que se ejecutará utilizando el transporte de WebSphere MQ Transport para SOAP. Se muestran los clientes Axis2 de ejemplo que se proporcionan con WebSphere MQ Transport para SOAP, y se utiliza el mandato **wsimport** para generar proxys.

Desarrollo de un cliente .NET 1 o 2 para el transporte WebSphere para SOAP utilizando Microsoft Visual Studio 2008
Desarrolle un cliente de servicio web .NET 1 o 2 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse

Desarrollar un cliente de servicio web Axis2 que se ejecutará utilizando el transporte de WebSphere MQ Transport para SOAP. Se muestran los clientes Axis2 de ejemplo que se proporcionan con WebSphere MQ Transport para SOAP, y se utiliza el mandato **wsimport** para generar proxys.

Antes de empezar

Obtenga las bibliotecas de Axis2 y configure un entorno de desarrollo y otro de prueba para poder ejecutar el cliente.

Nota: La denominación de las versiones y los releases que utiliza Axis crea confusión. Normalmente, Axis 1.4 hace referencia a la implementación de JAX-RPC, y Axis2 a la implementación JAX-WS.

Axis 1.4 es un nivel de versión. Si busca Axis 1.4 en Internet, se le sugerirá que visite http://ws.apache.org/axis/. La página contiene una lista de las versiones anteriores de Axis (1.2 y 1.3), y el último release de Axis 1.4, del 22 de abril de 2006. Hay versiones posteriores de Axis 1.4, que corrigen errores, pero se conocen como Axis 1.4. Es uno de estos releases de arreglo de errores que se entrega con WebSphere MQ. En el caso de Axis 1.4, utilice la versión del archivo `axis.jar` que se proporcione con WebSphere MQ, en lugar de la que se puede obtener de http://ws.apache.org/axis/.

El sitio web de Axis también hace referencia a Axis 1.1 para hacer referencia a todas las versiones de lo que, por lo general, se denomina Axis 1.4. Axis 1.2 se utiliza para hacer referencia a lo que normalmente se denomina Axis2.

Axis 1.5 no es un release posterior de Axis 1.4, es un release de Axis2. Si busca Axis 1.5 en Internet, se le sugerirá que visite http://ws.apache.org/axis2/.https://ws.apache.org/axis2/download.cgi contiene una lista de versiones de release de Axis2, con la etiqueta de 0.9 a 1.5.1 (e incluyendo, de forma confusa, la

versión 1.4). La versión de release de Axis2 que se debe utilizar con WebSphere Transport MQ para SOAP es la 1.4.1. Descargue Axis2 1.4.1 desde http://ws.apache.org/axis2/download/1_4_1/download.cgi.

Puede optar por generar proxys para los clientes de servicio web para WebSphere MQ Transport para SOAP utilizando el mandato **wsimport** o bien las herramientas que se proporcionan con un IDE. Eclipse IDE para Java EE Developer 3.5 SR1 utiliza **wsdl2java**.**wsimport** se proporciona con Java 6. Puede utilizar Java 5 para ejecutar proxies de cliente generados con **wsimport** o **wsdl2java**.

Los clientes Axis2 de ejemplo de servicio web que se proporcionan con WebSphere MQ Transport para SOAP se han desarrollado utilizando **wsimport**; consulte "Clientes Axis2 de ejemplo" en la página 1007.

La tarea siguiente muestra cómo generar y utilizar los proxies producidos por el asistente de servicios web empaquetado con Eclipse IDE para desarrolladores Java EE . En los clientes de ejemplo se muestra cómo utilizar los proxys que genera **wsimport**.

Para utilizar el asistente de servicios web, deberá añadir un servidor de aplicaciones que dé soporte a Axis2 en el entorno de trabajo. En los pasos se muestra cómo configurar WASCE para que dé soporte a Axis2, mediante el entorno de trabajo.

1. Configure el servidor de aplicaciones utilizado en Eclipse IDE para Java EE Developers para dar soporte a Axis2. En este ejemplo, configure WASCE 2.1.4, el servidor de aplicaciones, que forma parte del espacio de trabajo creado en "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980.
   a. Abra las preferencias del espacio de trabajo para configurar el servidor: abra **Ventana** > **Preferencias**.
   b. Compruebe que el JRE instalado sea `Java50`: Pulse **JRE instalados**.
   c. Añada WASCE como servidor: pulse **Servidor** > **Entornos de ejecución** > **Añadir ...** > **IBM** > **WASCE v2.1** > Siguiente. El JRE debe ser Java50 > Examinar hasta el directorio de instalación de WASCE > **Aceptar** > **Finalizar**. Debe haber instalado el plug-in WASCE para Eclipse Java EE IDE for Web Developers.
   d. Añada Axis2: Pulse **Servicios web** > **Axis2 Preferences**. En el separador **Axis2 Tiempo de ejecución** > **Examinar ...** Abra el directorio que contiene los muchos archivos jar de `Axis2` > **Apply**.
   e. Asocie WASCE con Axis2: Pulse **Servicios web** > **Server and Runtime**. En **`Server`** seleccione **IBM WASCE v2.1 Server**y, en **`Web service runtime`**, seleccione **Apache Axis2** > **Apply** > **OK**
   f. Iniciar el servidor: Abra la perspectiva Web y la vista Servidores. Pulse con el botón derecho del ratón en la vista Servidores > **Nuevo** > **Servidor**. **IBM WASCE v2.1 Server** está seleccionado y configurado > **Finalizar**. Inicie el servidor.
2. Compruebe que haya desplegado el servicio StockQuoteAxis en WASCE para poder ejecutar el asistente de servicio web.
3. Para probar el servicio con el servicio de WebSphere MQ Transport para SOAP, despliegue el servicio en un escucha de WebSphere MQ Transport para SOAP para Axis 1.4; consulte "Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse" en la página 980.

Acerca de esta tarea

El Eclipse IDE para Java EE Developers utiliza Java50 y el asistente de servicios web para generar las clases de proxy para el servicio. Las clases de proxy son diferentes de las clases creadas por la herramienta **wsimport** que se proporciona con Java 6. Un enfoque alternativo es generar las clases de proxy utilizando **wsimport** e importar los paquetes que crea en Eclipse Java EE IDE for Web Developers.

El asistente de servicios web en el Eclipse IDE para Java EE Developers crea un cliente de servicios web en un proyecto web. Puede ejecutar el cliente como una aplicación Java simple; no requiere un servidor de aplicaciones. También puede transferir el código a un proyecto Java y configurar la vía de acceso de construcción para incluir los archivos JAR Axis2 .

Procedimiento

1. Cree un proyecto web en un nuevo proyecto de Enterprise:
   a) Sin nada seleccionado en el Explorador de proyectos > Pulse con el botón derecho del ratón en el espacio en blanco > **Nuevo** > **Proyecto de aplicación de empresa** > Nombrarlo `StockQuoteAxis2EAR` > **Finalizar**. Responda No a la ventana dándole la opción de abrir la perspectiva Java EE .

      Los valores predeterminados están establecidos para utilizar WASCE.

   b) Pulse con el botón derecho `StockQuoteAxis2EAR` > **Nuevo** > **Proyecto web dinámico**. Denomine el proyecto `StockQuoteAxis2WebClient` > Compruebe el recuadro de pertenencia EAR para añadir el proyecto a **StockQuoteAxis2EAR**. Se selecciona WASCE 2.1 como tiempo de ejecución de destino.

   c) En la sección Configuración de la página **Nuevo proyecto web dinámico** > **Modificar ...** > Compruebe la faceta de proyecto de servicios web Axis2 . **Módulo web dinámico 2.5**, **Java 6.0**y **WASCE deployment 1.2** ya están seleccionados. > **Aceptar** > **Finalizar**. Responda No a la ventana dándole la opción de abrir la perspectiva Java EE .

2. Importe WSDL para el servicio en el espacio de trabajo y genere el proxy del cliente:

   En este ejemplo, el documento WSDL contiene el vínculo del servicio HTTP y se convierte en el destino para el proxy del cliente web estático. Puede modificar el URL en el vínculo del servicio web de modo que apunte al URL de WebSphere MQ Transport para SOAP antes de generar el proxy del cliente. El proxy del cliente web estático es, por tanto, el servicio que se despliega en WebSphere MQ Transport para SOAP.

   a) Inicie el Explorador de servicios web: utilice el icono de la barra de acciones o **Ejecutar** > **Iniciar el Explorador de servicios web**.

   b) Seleccione el explorador WSDL pulsando el icono WSDL en la ventana **Explorador de servicios web** > Pulse **WSDL principal** en la ventana Navigator > Escriba el URL del archivo WSDL del eje StockQuote> **Ir**.
   En este ejemplo, obtenga el WSDL directamente del servicio HTTP: `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`

   c) En el Navegador, pulse la línea en la que aparezca el URL del servicio web. En la ventana **Acciones** , pulse **Importar WSDL a entorno de trabajo** > Seleccionar un **StockQuoteAxis2WebClient** como **Proyecto de entorno de trabajo** > Escriba el **nombre de archivo WSDL**, `StockQuoteAxisHTTP.wsdl` > **Ir**.

   d) Pulse con el botón derecho del ratón en **StockQuoteAxisHTTP.wsdl** > **Servicios web** > **Generar cliente**. Compruebe que la información de configuración que aparece en la página de servicios web del asistente sea la siguiente: Servidor: IBM WASCE v2.1 Server, Tiempo de ejecución de servicio web: Apache Axis2, Proyecto del cliente: StockQuoteAxis2WebClient, Proyecto EAR del cliente: StockQuoteAxisEAR. Para corregir la configuración, pulse en las líneas que sean incorrectas.

   e) Pulse **Siguiente** > verificar los valores de generación de código > **Finalizar**.

      Tenga en cuenta que se crea un nuevo paquete, `soap.server`, que contiene los proxys que necesita.

3. Configure el proyecto para que ejecute WebSphere MQ Transport para SOAP como transporte JMS.

   WebSphere MQ Transport para SOAP proporciona un `transportSender`, pero no ningún `transportReceiver`. En otras palabras, WebSphere MQ Transport para SOAP da soporte a los clientes Axis2. Actualmente no da soporte a los servicios de Axis2.

   a) En el proyecto **StockQuoteAxis2WebClient** , pulse con el botón derecho `WebContent\WEB-INF\conf\axis2.xml` > **Abrir con ...** > **Editor XML**.

   b) Busque el último `transportSender` (hacia el final del archivo) y busque el JMS comentado `transportSender` > Pulse con el botón derecho del ratón en la línea > **Añadir antes de ...** > **transportSender**.

   c) Pulse con el botón derecho **transportSender** > **Añadir atributo** > **Nombre** > Pulse con el botón derecho **transportSender** > **Añadir atributo** > **Clase**.

d) Pulse con el botón derecho del ratón en **Nombre** > **Editar atributo** > Escriba el valor de **:** `jms`

e) Pulse con el botón derecho **Clase** > **Editar atributo** > Escriba el **Valor:** `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`. > Guardar.

f) Añada `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` a la vía de acceso de compilación: pulse con el botón derecho del ratón en **StockQuoteAxis2WebClient** > **Vía de acceso de compilación** > **Configurar vía de acceso de compilación ...** > Pulse la pestaña **Bibliotecas** > **Añadir JAR externos ...**. Seleccione todos los JAR en *MQ_INSTALLATION_PATH*`\java\lib` > **Aceptar**.

MQ_INSTALLATION_PATH es el directorio donde está instalado WebSphere MQ.

4. Cree un cliente estático síncrono, pruébelo utilizando HTTP y, a continuación, convierta el proxy para ejecutar el cliente estático utilizando WebSphere MQ Transport para SOAP.

a) Pulse con el botón derecho **Recursos Java: src** > **Nuevo** > **Paquete** > Nombrar el paquete `soap.client` > Finalizar

b) Pulse con el botón derecho del ratón en **soap.client** > **New** > **Class** > Name the class `SQA2StaticClient` > **Finish**.

c) Sustituya la clase por el código siguiente y, a continuación, pulse **Guardar**.

Figura 182. SQA2DynamicClient.java

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. Pruebe el cliente con el servicio StockQuoteAxis desplegado en WASCE, y con WebSphere MQ Transport para SOAP.

a) En el Explorador de proyectos, pulse con el botón derecho del ratón en **SQA2StaticClient** > **Ejecutar como ...** > **Aplicación Java**.

El resultado, `Response is 55.25`, aparece en la vista Consola. También puede seleccionar la ventana de consola WASCE en la vista Consola y ver la salida en el servidor WASCE, `StockQuoteAxis called with parameter: ibm`.

b) El proxy se ha creado con la dirección de servicio, `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis`, por lo que el cliente estático llama al servicio que se ejecuta en HTTP. Puede cambiar el cliente estático para invoque al servicio mediante WebSphere MQ Transport para SOAP. Las siguientes instrucciones sirven para cambiar la dirección de servicio en `StockQuoteAxisServiceStub.java` sin volver a crear el proxy, y configurar los parámetros de tiempo de ejecución de `SQA2StaticClient` para que se cargue `axis2.xml`. Puede configurar `axis2.xml` que configura Axis2 para que utilice WebSphere MQ Transport para SOAP.

c) Abra `StockQuoteAxisServiceStub.java` >
Sustituya las dos apariciones de `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` por,

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

```
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

d) Si ejecuta SQA2StaticClient ahora, se genera una excepción porque no se ha encontrado ningún transportSender que esté configurado para JMS.
La excepción es:

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) En el Explorador de proyectos, pulse con el botón derecho del ratón en **SQA2StaticClient** > **Ejecutar como ...** > **Ejecutar configuraciones ...**. Cambie al separador **(x) = Argumentos** y, en el área de entrada **Argumentos de VM** , escriba la vía de acceso al archivo axis2.conf > **Aplicar** > **Ejecutar**.
El argumento de VM es: -Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml. O puede proporcionar una vía de acceso estándar del archivo de configuración de Axis2.

f) Vuelva a ejecutar SQA2StaticClient. En esta ejecución, está utilizando WebSphere MQ Transport para SOAP. Confírmelo comprobando que no haya ninguna nueva salida en la consola WASCE. Abra la consola o ventana de mandatos que está asociada con el escucha SimpleJavay la salida allí es StockQuoteAxis called with parameter: ibm.

6. Cree un cliente dinámico para HTTP y WebSphere MQ Transport para SOAP, y pruébelo.

a) Pulse con el botón derecho del ratón en **soap.client** > **New** > **Class** > Name the class SQA2DynamicClient > **Finish**.

b) Sustituya la clase por el código siguiente y, a continuación, pulse **Guardar**.

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
     } catch (Exception e) {
       System.out.println("Exception: " + e.getMessage());
       e.printStackTrace();
     }
   }
 }
```

c) Cree una configuración de ejecución para SQA2DynamicClient.java, y añada la vía de acceso a axis2.xml:
-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml

d) Ejecute SQA2DynamicClient. Consulte la consola de salida de SQA2DynamicClient WASCE y **SimpleJavaListener**.

7. Cree un cliente asíncrono, y acceda al resultado en un manejador de devolución de llamada, y en la hebra del programa principal.

Los proxies de cliente asíncrono creados por el asistente de servicios web para Eclipse Java EE IDE for Web Developers difieren de los proxies creados por **wsimport**. Los proxys de **wsimport** utilizan los tipos genéricos Future, Response y AsyncHandler.

El asistente de servicios web para Eclipse Java EE IDE for Web Developers crea una clase abstracta `StockQuoteAxisServiceCallbackHandler` . Debe ampliar `StockQuoteAxisServiceCallbackHandler` y crear un manejador de devolución de llamada.

a) Pulse con el botón derecho del ratón en **soap.client** > **New** > **Class** > Name the class `SQA2CallbackHandler` > **Finish**.

b) Sustituya la clase por el código siguiente.

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) Pulse con el botón derecho del ratón en **soap.client** > **New** > **Class** > Name the class `SQA2AsyncClient` > **Finish**.

d) Sustituya la clase por el código siguiente.

Figura 183. `SQA2AsyncClient.java`

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
```

```
            e.printStackTrace();
        }
      }
    }
```

La salida de la consola es la siguiente:

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

Clientes Axis2 de ejemplo

Los proxies de ejemplo se generan utilizando la herramienta **wsimport** que se empaqueta con Java 6. Se proporcionan seis muestras:

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

Los ejemplos de cliente se generan para el servidor StockQuoteAxis de ejemplo. Genere el WSDL con el mandato **amqwdepoyWMQServer**, especificando el conmutador -w para seleccionar el estilo rpcLiteral. Utilice el mandato siguiente para generar los proxys para los ejemplos:

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

Figura 184. DynamicProxyClientSync.java

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
```

```
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 185. DynamicProxyClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by
polling...");
      Response<Float> response = service.getQuoteAsync("49");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
```

```
    } // end of catch block
  }
}
```

Figura 186. DynamicProxyClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());
```

```
        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Figura 187. DispatchClientSync.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");
```

```
      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
      System.out.println(" > getQuoteOneWay call has returned");

      /****************************************
       * Create Request Reply SOAPMessage request.
       ****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 188. DispatchClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by
polling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");
```

```
      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 189. DispatchClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");
```

```
      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
```

```
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Tareas relacionadas

Desarrollo de un cliente JAX-RPC para el transporte WebSphere para SOAP utilizando Eclipse
Desarrolle un cliente de servicio web Axis 1.4 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Desarrollo de un cliente .NET 1 o 2 para el transporte WebSphere para SOAP utilizando Microsoft Visual Studio 2008
Desarrolle un cliente de servicio web .NET 1 o 2 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Desarrollo de un cliente .NET 1 o 2 para el transporte WebSphere para SOAP utilizando Microsoft Visual Studio 2008

Desarrolle un cliente de servicio web .NET 1 o 2 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Antes de empezar

Puede iniciar el desarrollo de un cliente .NET 1 o 2 de varias maneras diferentes:

1. Utilice **amqwdeployWMQService** para generar apéndices de cliente desde un servicio web e importarlos a Visual Studio.
2. Utilice **java2wsdl** para generar WSDL a partir de una implementación Java de un servicio web y, a continuación, utilice `wsdl.exe`, que se suministra con .NET, para generar apéndices de cliente.
3. Genere WSDL a partir de una implementación .NET `.asmx` del servicio utilizando **amqswsdl**y, a continuación, utilice `wsdl.exe`.
4. Si ha desarrollado y desplegado el servicio para HTTP, utilice **Añadir referencia web ...** asistente en Visual Studio para configurar el cliente para acceder al servicio HTTP. Modifique el URL para el servicio desplegado en el transporte WebSphere MQ para SOAP.

La tarea utiliza el servicio desarrollado en “Desarrollo de un servicio .NET 1 o 2 para el transporte WebSphere MQ para SOAP utilizando Microsoft Visual Studio 2008” en la página 984.

Acerca de esta tarea

Siga estos pasos para crear un cliente .NET 1 o 2 para HTTP y un transporte WebSphere MQ para SOAP.

Procedimiento

1. Cree la aplicación de consola de cliente y modifíquela para invocar el servicio web HTTP StockQuote .
   a) Pulse con el botón derecho **Solución 'StockQuoteDotNet'** en el **Explorador de soluciones** > Añadir ... > Nuevo proyecto. Seleccione el tipo de proyecto **C#** , **.NET Framework 2.0**y **Aplicación de consola**. Asigne al proyecto el nombre `StockQuoteClientDotNet` > **Aceptar**
   b) Pulse con el botón derecho **Solución 'StockQuoteDotNet'** en el **Explorador de soluciones** > Añadir ... > Nuevo proyecto. Seleccione el tipo de proyecto **C#** , **.NET Framework 2.0**y **Aplicación de consola**. Asigne al proyecto el nombre `StockQuoteClientDotNet` > **Aceptar**
   c) Pulse con el botón derecho del ratón en **StockQuoteClientDotNet** > **Establecer como proyecto de inicio**.
   d) Pulse con el botón derecho **StockQuoteClientDotNet** > **Añadir referencia web ...** > Examinar a servicios web en esta solución > Seleccionar **StockQuote** > **Añadir referencia**. Observe que ha añadido una referencia web al host local y un nuevo archivo de configuración `app.config`.
   e) En el Explorador de soluciones, cambie el nombre de la aplicación de consola de `Program.cs` a `StockQuoteClientDotNet.cs` > Pulse **Aceptar** para cambiar todos los usos de `Program.cs` a `StockQuoteClientDotNet.cs`.
   f) Sustituya el contenido de `StockQuoteClientDotNet.cs` por el código en Figura 190 en la página 1016.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figura 190. Programa HTTP StockQuoteClientDotNet

   g) Inicie StockQuoteClientDotNet para probar el servicio `StockQuote.asmx` :
      i) Pulse **F5**, pulse la flecha verde en la barra de acciones o **Depurar** > **Iniciar depuración (F5)**.

      Si el proyecto StockQuoteDOtNet está en la misma solución, se inicia automáticamente. De lo contrario, primero debe iniciar el servicio.

      La ventana de mandatos con los resultados se abre detrás del espacio de trabajo. La sentencia `Console.ReadLine();` impide que se cierre hasta que pulse **Intro**.

   Consejo: Asegúrese de que `StockQuote.asmx` es la página de inicio del proyecto `StockQuoteDotNet` .
2. Modifique StockQuoteClientDotNet para llamar al servicio `StockQuote.asmx` utilizando el transporte WebSphere MQ para SOAP.
   a) Añada las líneas que se muestran en negrita al cliente.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figura 191. Programa StockQuoteClientDotNet modificado

De forma alternativa, modifique el URL predeterminado. Abra **StockQuoteClientDotNet** > **Properties** > **Settings.settings** y cambie el valor de la propiedad `StockQuoteClientDotNet_localhost_StockQuote` por el transporte WebSphere MQ para el URL SOAP.

b) Añadir una referencia a `amqsoap.dll`

i) En el proyecto **StockQuoteClientDotNet** en el **Explorador de soluciones**, pulse con el botón derecho **Referencias** > **Añadir referencia ...** > Pulse la pestaña **Examinar** > vaya a *MQ_INSTALLATION_PATH*`\bin` > Seleccione **amqsoap.dll** > **Aceptar**. *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ .

3. Pruebe el cliente con el servicio `StockQuote.asmx` utilizando el transporte WebSphere MQ para SOAP.

a) Abra una ventana de mandatos en el directorio de proyecto StockQuoteDotNet : `.\StockQuoteDotNet\StockQuoteDotNet` > Verifique que `.bin\StockQuoteDotNet.dll` existe. Si no lo hace, vuelva a crear la solución.

b) Escriba el mandato **amqwRegisterdotNet**.

Sólo necesita ejecutar **amqwRegisterdotNet** una vez por instalación.

c) Si ha ejecutado **amqwdeployWMQServer** con `genAsmxWMQBits`, ejecute el escucha SOAP de .NET: `generated\server\startWMQNListener`

d) De forma alternativa, ejecute el escucha directamente:

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. En Visual Studio 2008, pulse **F5** para ejecutar StockQuoteClientDotNet.

Clientes de servicios web .NET Framework 1 y .NET Framework 2

Los clientes .NET de ejemplo proporcionados con el transporte WebSphere MQ para SOAP utilizan apéndices generados para llamar a los servicios Axis y .NET de ejemplo.

Para clientes .NET Framework 1 y .NET Framework 2, WebSphere MQ proporciona acceso a servicios web utilizando clientes .NET. El mandato **amqwdeployWMQService** tiene una opción,

genProxiestoDotNet, que genera apéndices de cliente .NET Framework 1 o .NET Framework 2 para un servicio web. También puede utilizar apéndices de cliente generados por la herramienta .NET **wsdl** , o por Microsoft Visual Studio 2005 o 2008.

Los clientes de servicio .NET Framework 1 y .NET Web de ejemplo se instalan en *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet. *MQ_INSTALLATION_PATH* es el directorio donde está instalado WebSphere MQ .

SQVB2Axis.vb
SQVB2Axis.vb, Figura 192 en la página 1018, es el cliente de Visual Basic para llamar al servicio **StockQuoteAxisService** .

SQVB2DotNet.vb
QVB2DotNet.vb, Figura 193 en la página 1018, es el cliente de Visual Basic para llamar al servicio **StockQuoteDotNet** .

SQCS2Axis.cs
SQCS2Axis.cs, Figura 194 en la página 1019, es el cliente C# para llamar al servicio **StockQuoteAxisService** . Puede alterar temporalmente el URL del servicio proporcionando un URL en la línea de mandatos.

SQCS2DotNet.cs
SQCS2DotNet.cs, Figura 195 en la página 1019, es el cliente C# para llamar al servicio **StockQuoteDotNet** . Puede alterar temporalmente el URL del servicio proporcionando un URL en la línea de mandatos.

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

Figura 192. SQVB2Axis

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

Figura 193. SQVB2DotNet

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figura 194. SQCS2Axis

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figura 195. SQCS2DotNet

Tareas relacionadas

Desarrollo de un cliente JAX-RPC para el transporte WebSphere para SOAP utilizando Eclipse
Desarrolle un cliente de servicio web Axis 1.4 para ejecutarlo utilizando el transporte WebSphere MQ para SOAP.

Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse
Desarrollar un cliente de servicio web Axis2 que se ejecutará utilizando el transporte de WebSphere MQ Transport para SOAP. Se muestran los clientes Axis2 de ejemplo que se proporcionan con WebSphere MQ Transport para SOAP, y se utiliza el mandato **wsimport** para generar proxys.

Despliegue de servicios web utilizando el transporte WebSphere MQ para SOAP

Despliegue un servicio web en uno de varios entornos de servidor diferentes y conéctese a él utilizando el transporte WebSphere MQ para SOAP.

Antes de empezar

Desarrolle un servicio web y pruébelo utilizando SOAP sobre HTTP en el entorno de destino.

Acerca de esta tarea

Puede desplegar un servicio web para que se ejecute con el transporte WebSphere MQ para SOAP en varios entornos de tiempo de ejecución SOAP diferentes. Puede desplegar un servicio en Axis 1.4 utilizando solo el software instalado con WebSphere MQ. Para los otros entornos de tiempo de ejecución, debe instalar software adicional.

No está restringido a ejecutar el transporte WebSphere MQ para SOAP a los servidores para los que hay instrucciones de despliegue. Utilice las instrucciones para desplegar un servicio en uno de los entornos listados.

Nota: Algunos entornos integrados ofrecen SOAP sobre JMS utilizando el enlace SOAP JMS recomendado W3C , así como el transporte WebSphere MQ para el enlace SOAP. Los releases de WebSphere MQ, hasta 7.0.1.2inclusive, sólo dan soporte al transporte WebSphere MQ para el enlace SOAP. A partir de 7.0.1.3 , puede desplegar clientes Axis2 utilizando un URI que se ajuste a la recomendación de candidato W3C para SOAP sobre JMS. Consulte la guía de aprendizaje Desarrollo de una aplicación de servicios web JAX-WS SOAP/JMS con WebSphere Application Server V7 y Rational Application Developer V7.5.

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService

Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Antes de empezar

1. Siga las instrucciones para instalar WebSphere MQ Transport para SOAP
2. Verifique la instalación y el entorno utilizando el mandato **runivt** .
3. Para volver a desplegar un servicio:
   a. Suprima el subdirectorio `./generated` y todos sus subdirectorios.
   b. Elimine las solicitudes de la cola de destino y suprímalas.
   c. Continúe con las instrucciones del paso “2” en la página 1020.

Acerca de esta tarea

Estas instrucciones son para desplegar un servicio Axis 1.4 por primera vez. Para reiniciar un servicio Axis 1.4 , vuelva a ejecutar el escucha SOAP de Axis 1.4 : paso “11” en la página 1021.

Utilice las instrucciones siguientes para desplegar un nuevo servicio Axis 1.4 en el transporte WebSphere MQ para SOAP:

Procedimiento

1. Cree un directorio *deployDir* para contener los archivos de despliegue.
   El programa de utilidad de despliegue requiere que cada servicio se despliegue desde un directorio distinto.
2. Abra una ventana de mandatos en Windows, o un shell de mandatos utilizando X Window System en sistemas UNIX and Linux , en *deployDir* para ejecutar **amqwdeployWMQService**.
3. Ejecute **amqwsetcp** para establecer la vía de acceso de clases.
   JRE y JDK deben estar en la vía de acceso de clases, en la versión 5.0 o posterior, y en el mismo nivel de versión.
4. Copie el origen de clase, *className*`.java`, en *deployDir*
5. Copie todos los archivos de origen Java en el mismo paquete que *className* en *deployDir*`/`*packageName*, donde *packageName* es un árbol de directorios correspondiente al nombre del paquete.
6. Ejecute **javac** *packageName*`.`*className*`.`

Es posible que tenga que añadir una vía de acceso al directorio actual ".", o al directorio *packageName* para **javac** para encontrar las otras clases.

7. Cree el WSDL Axis para el servicio:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. Cree los recursos de WebSphere MQ para el servicio:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

Consejo:

Si desea configurar un nuevo gestor de colas y los recursos que necesita, para realizar el desarrollo y las pruebas, ejecute **setupWMQSOAP**.

Si desea configurar el nuevo gestor de colas como valor predeterminado, realice una copia de **setupWMQSOAP** desde el directorio *WMQ install directory*\tools\soap\samples y añada el parámetro -q a la línea

```
call :try -q crtmqm %QMGR%
```

9. Cree el escucha Axis y despliegue el servicio:

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. Si necesita generar el WSDL para el servicio, generar apéndices de cliente o proxies de cliente, ejecute **amqwdeployWMQService** con uno de los parámetros siguientes:

    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - genProxiestoEje

    Nota: Debe generar WSDL antes de generar los proxies. La opción AllAxis falla si CLASSPATH no está configurada para encontrar todas las clases que se importan para compilar *className*.java. Si hay varios archivos Java en el paquete que contiene *className*.java, primero debe compilarlos utilizando **javac**. **amqwdeployWMQService** -f *packageName*.*className*.java -c CompileJava sólo compila *className*.java.

11. Inicie el escucha Axis generado.

```
.\generated\server\startWMQJListener.cmd
```

Tareas relacionadas

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP
Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP
El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP
WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS

Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP
El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP

Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Antes de empezar

1. Siga las instrucciones para instalar WebSphere MQ Transport para SOAP
2. Verifique la instalación y el entorno utilizando el mandato **runivt** .
3. Se debe establecer la vía de acceso a los archivos de .NET Framework `wsdl.exe` y `csc.exe` . Las copias de `wsdl.exe` y `csc.exe` identificadas por la variable PATH deben estar en el mismo nivel que .NET Framework. Si tiene varias infraestructuras .NET instaladas o está utilizando Visual Studio, compruebe cuidadosamente la variable PATH .
4. Para volver a desplegar un servicio:
   a. Suprimir el subdirectorio `./generated` y todos sus subdirectorios
   b. Elimine las solicitudes de la cola de destino y suprímalas.
   c. Continúe con las instrucciones del paso “2” en la página 1022.

Acerca de esta tarea

Estas instrucciones son para desplegar un servicio .NET por primera vez. Para reiniciar un servicio .NET, vuelva a ejecutar el escucha SOAP .NET, paso “9” en la página 1023.

Utilice las instrucciones siguientes para desplegar un nuevo servicio .NET Framework 1 o .NET Framework 2 en el transporte WebSphere MQ para SOAP:

Procedimiento

1. Cree un directorio *deployDir* para contener los archivos de despliegue.
   El programa de utilidad de despliegue requiere que cada servicio se despliegue desde un directorio distinto.
2. Abra una ventana de mandatos en *deployDir* para ejecutar **amqwdeployWMQService**.

```
C:\IBM\ID\QuoteClient>
```

3. Ejecute **amqwsetcp** para establecer la vía de acceso de clases.
   Sólo se necesita una vía de acceso de clases para los clientes Axis.
4. Copie el servicio .NET, *className*`.asmx`, en *deployDir*
5. Cree la implementación de servicio en una biblioteca (`.dll`).
   La implementación del servicio en línea se encuentra en *className*`.asmx`. La implementación del servicio de detrás de código puede ser *className*`.asmx.cs`.
   La Figura 196 en la página 1023 muestra un ejemplo de un mandato para crear un servicio V2 de .NET Framework como una biblioteca.

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

Figura 196. Mandato de compilación para el servicio V2 de .NET Framework

6. Copie *className*.dll en *deployDir**bin*.
7. Configure los recursos de WebSphere MQ y cree el escucha necesario para el servicio:

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. Si necesita generar el WSDL para el servicio, generar apéndices de cliente o proxies de cliente, ejecute **amqwdeployWMQService** con uno de los parámetros siguientes:
   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - genProxiestoEje

   Nota: Debe generar WSDL antes de generar los proxies.
9. Inicie el escucha .NET generado.

```
.\generated\server\startWMQNListener.cmd
```

Tareas relacionadas

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService
Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP
El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP
WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP
El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP

El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Antes de empezar

Utilice las mismas herramientas para desarrollar para un cliente o servicio para WebSphere MQ, como desarrollaría para HTTP. CICS tiene herramientas correspondientes a **Java2wsdl** y **wsdl2Java**:

- **DFHWS2LS** toma una descripción de servicio web como punto de partida. Utiliza las descripciones de los mensajes, y los tipos de datos utilizados en dichos mensajes, para construir estructuras de datos de lenguaje de alto nivel. Puede utilizar en las estructuras de los programas de aplicación escritos en distintos idiomas.
- **DFHLS2WS** toma una estructura de datos de lenguaje de alto nivel como punto de partida. Utiliza la estructura para construir una descripción de servicios web que contiene descripciones de mensajes. También crea esquemas para los mensajes de la estructura de datos de lenguaje.

Siga las instrucciones, Creación de un servicio web en la documentación del producto CICS , para crear un servicio web.

Acerca de esta tarea

Siga las instrucciones, Configuración de CICS para utilizar el transporte WebSphere MQ en la documentación del producto CICS . Utilizando las instrucciones, puede desplegar el servicio web en el transporte WebSphere MQ para SOAP.

Tareas relacionadas

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService
Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP
Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP
WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP
El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP

WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Antes de empezar

Utilice Rational Application Developer, WebSphere Integration Developer o un kit de herramientas de servicios web para desarrollar el servicio web.

Acerca de esta tarea

Utilice las instrucciones siguientes para desplegar un servicio utilizando el transporte WebSphere MQ para SOAP como transporte SOAP en WebSphere Application Server.

Procedimiento

1. Configure WebSphere MQ como proveedor de mensajería JMS para el bus de integración de servicios en WebSphere Application Server.
2. Configure los recursos de WebSphere MQ necesarios para el servicio.
3. Siga las instrucciones, Configuración de recursos JMS para el escucha de punto final SOAP sobre JMS síncrono, en la documentación del producto WebSphere Application Server Network Deployment.

   Hay instrucciones correspondientes para otras plataformas WebSphere Application Server.
4. Modifique el URI de servicio para que se ajuste al transporte de WebSphere MQ para URI SOAP.
5. Despliegue el servicio en WebSphere Application Server.

Qué hacer a continuación

Despliegue el servicio con un transporte HTTP para que los clientes puedan consultar el servicio y recibir el WSDL en respuesta.

Tareas relacionadas

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService
Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP
Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP
El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP
El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS

Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Antes de empezar

La tarea requiere WebSphere Application Server v7.0.0.9 y WebSphere MQ v7.0.1.3.

Acerca de esta tarea

La tarea tiene dos pasos:

Procedimiento

1. “Configurar recursos de WebSphere MQ” en la página 1026
2. “Configurar recursos de WebSphere Application Server” en la página 1027

Qué hacer a continuación

“Configurar recursos de WebSphere MQ” en la página 1026

Tareas relacionadas

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService
Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP
Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP
El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP
WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP
El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Configurar recursos de WebSphere MQ

Antes de empezar

Para el soporte de Axis2 , necesita WebSphere MQ 7.0.1.3 o posterior.

Acerca de esta tarea

Para mayor simplicidad, la tarea presupone que WebSphere MQ está instalado en la misma estación de trabajo que el otro software y utiliza conexiones de enlaces. Las configuraciones de cliente de WebSphere Application Server y Axis2 funcionan con conexiones de cliente. Para ejecutar con la tarea utilizando conexiones de cliente, compruebe que puede colocar y obtener mensajes hacia y desde las colas de solicitudes y respuestas del cliente Axis2 y de los sistemas WebSphere Application Server.

De nuevo, por simplicidad, no se utiliza ninguna configuración de seguridad. El ID de usuario tiene autorización `mqm` completa.

Procedimiento

1. Cree un gestor de colas predeterminado, `QM1`.

   Utilice WebSphere MQ Explorer para crear `QM1` como gestor de colas predeterminado. Configúrelo para que se inicie automáticamente y seleccione la opción para crear un escucha. De forma alternativa, utilice los mandatos siguientes:

```
crtmqm -q -sa QM1
strmqm
echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
           control(qmgr) replace | runmqsc
echo start listener(LISTENER.TCP) | runmqsc
```

2. Defina una cola de solicitudes, REQUESTAXIS, y una cola de respuestas, REPLYAXIS.

   Utilice el Explorador o los mandatos siguientes:

```
echo define ql(REQUESTAXIS) replace | runmqsc
echo define ql(REPLYAXIS) replace | runmqsc
```

Qué hacer a continuación

"Configurar recursos de WebSphere Application Server" en la página 1027

Configurar recursos de WebSphere Application Server

Antes de empezar

Para el soporte de W3C SOAP sobre JMS, necesita WebSphere Application Server v7. Esta configuración se ha realizado en WebSphere Application Server Versión 7.0 Test Environment v7.0.0.9 Actualización 1. WebSphere Application Server se ha proporcionado con Rational Software Architect for WebSphere Software 7.5.4. Rational Software Architect se ha actualizado a v7.5.5.1, aplicando las últimas actualizaciones disponibles.

Como parte del proceso de instalación, cree un perfil para WebSphere Application Server. En la tarea, la seguridad administrativa no está habilitada. El nombre de perfil predeterminado es `was70profile1`y el servidor es `server1`.

Acerca de esta tarea

Configure WebSphere Application Server. Puede iniciar el servidor desde Rational Application Developer, e iniciar la consola administrativa desde la vista Servidores, o puede iniciar el servidor utilizando un archivo de mandatos y administrar el servidor utilizando un navegador. La tarea utiliza el segundo método.

Los archivos de mandatos del servidor están en la carpeta, *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\bin`. Los archivos de registro que puede inspeccionar se encuentran en *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\logs\server1`.

Como convenio, todos los nombres de objeto WebSphere MQ están en mayúsculas y todos los nombres JNDI que hacen referencia a los objetos WebSphere MQ están en minúsculas.

Procedimiento

1. Inicie el servidor.

```
startServer server1
```

2. Inicie un navegador, abra la consola de administración e inicie sesión.

```
http://localhost:9061/ibm/console/unsecureLogon.jsp
```

   Escriba cualquier serie en el campo de ID de usuario.

3. Crear una fábrica de conexiones, `qm1`
   a) En el navegador, abra **Recursos** > **JMS** > **Fábricas de conexiones**.
   b) En la ventana Fábricas de conexiones, seleccione el ámbito **Nodo =*nombre_nodo***y pulse **Nuevo**.
   c) Seleccione **WebSphere MQ** > **Aceptar**.

d) Proporcione la información de conexión del gestor de colas desde Tabla 143 en la página 1028 > **Siguiente**.

Tabla 143. Información de conexión del gestor de colas	
Nombre de campo	**Valor**
Nombre	qm1
Nombre JNDI	qm1

e) Seleccione **Especificar toda la información necesaria en este asistente** como método de conexión > **Siguiente**.

f) Escriba QM1 como detalles de conexión de cola > **Siguiente**.

g) Especifique los detalles de conexión en Tabla 144 en la página 1028 > **Siguiente**.

Tabla 144. Detalles de conexión	
Nombre de campo	**Valor**
Transporte	Bindings, then client
hostname	localhost
Puerto	1414
Canal de conexión de servidor	SYSTEM.DEF.SVRCONN

h) **Probar conexión** > **Siguiente** > **Finalizar** > **Guardar**.

4. Cree la cola de solicitudes JMS, requestaxis.

a) En el Navigator, abra **Recursos** > **JMS** > **Colas**.

b) En la ventana Fábricas de conexiones, seleccione el ámbito **Nodo =*nombre_nodo***y pulse **Nuevo**.

c) Seleccione **WebSphere MQ** > **Aceptar**.

d) Especifique los detalles de cola en Tabla 145 en la página 1028 > **Aceptar** > **Guardar**.

Tabla 145. Detalles de cola	
Nombre de campo	**Valor**
Nombre	requestaxis
Nombre JNDI	requestaxis
Nombre de cola	REQUESTAXIS
Nombre del gestor de colas	QM1

5. Repita el paso "4" en la página 1028 para crear la cola de respuestas JMS, replyaxis.

6. Cree una especificación de activación, qm1as.

La especificación de activación desencadena el MDB (Message Driven Bean) del direccionador de servicios web cuando llega un mensaje a la cola de solicitudes. El MDB se define en el descriptor de despliegue del servicio web creado por el asistente de servicios web de Rational Application Developer.

a) En el Navigator, abra **Recursos** > **JMS** > **Especificaciones de activación**.

b) En la ventana Fábricas de conexiones, seleccione el ámbito **Nodo =*nombre_nodo***y pulse **Nuevo**.

c) Seleccione **WebSphere MQ** > **Aceptar**.

d) Especifique los atributos básicos de la especificación de activación desde Tabla 146 en la página 1029 > **Siguiente**.

Tabla 146. Nombre de especificaciones de activación	
Nombre de campo	**Valor**
Nombre	`qm1as`
Nombre JNDI	`qm1as`

e) Especifique su información de MDB desde Tabla 147 en la página 1029 > **Siguiente**.

Tabla 147. Información de MDB	
Nombre de campo	**Valor**
Nombre NDI de destino	`requestaxis`
Selector de mensajes	*Se ha dejado en blanco*
Tipo de destino	`Queue`

f) Seleccione **Especificar toda la información necesaria en este asistente** como método de conexión > **Siguiente**.

g) Escriba `QM1` como detalles de conexión de cola > **Siguiente**.

h) Especifique los detalles de conexión en Tabla 144 en la página 1028 > **Siguiente**.

Tabla 148. Detalles de conexión	
Nombre de campo	**Valor**
Transporte	`Bindings, then client`
hostname	`localhost`
Puerto	`1414`
Canal de conexión de servidor	`SYSTEM.DEF.SVRCONN`

i) **Probar conexión** > **Siguiente** > **Finalizar** > **Guardar**.

7. Cree una fábrica de conexiones de cola, `jms/WebServicesReplyQCF`, para la cola de respuestas.

   El direccionador de servicios web utiliza una fábrica de conexiones de cola para acceder a una cola de respuestas. En el descriptor de despliegue del servicio web, a la fábrica de conexiones de cola se le asigna el nombre JNDI predeterminado `jms/WebServicesReplyQCF`. Puede cambiar el nombre en el descriptor de despliegue. En esta tarea, añada el nombre predeterminado a las definiciones de recursos JMS.

   a) En el Navigator, abra **Recursos** > **JMS** > **Fábricas de conexiones de cola**.

   b) En la ventana Fábricas de conexiones, seleccione el ámbito **Nodo =*nombre_nodo***y pulse **Nuevo**.

   c) Seleccione **WebSphere MQ** > **Aceptar**.

   d) Especifique los atributos básicos de la fábrica de conexiones de cola desde Tabla 149 en la página 1029 > **Siguiente**.

Tabla 149. Nombre de fábrica de conexiones de cola	
Nombre de campo	**Valor**
Nombre	`WebServicesReplyQCF`
Nombre JNDI	`jms/WebServicesReplyQCF`

   e) Seleccione **Especificar toda la información necesaria en este asistente** como método de conexión > **Siguiente**.

   f) Escriba `QM1` como detalles de conexión de cola > **Siguiente**.

   g) Especifique los detalles de conexión en Tabla 144 en la página 1028 > **Siguiente**.

Tabla 150. Detalles de conexión	
Nombre de campo	**Valor**
Transporte	`Bindings, then client`
hostname	`localhost`
Puerto	1414
Canal de conexión de servidor	`SYSTEM.DEF.SVRCONN`

h) **Probar conexión** > **Siguiente** > **Finalizar** > **Guardar**.

Qué hacer a continuación

"Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS" en la página 987

Despliegue de un servicio en el punto final de servicio de WebSphere ESB y Process Server para utilizar WebSphere Transport for SOAP

El transporte WebSphere MQ para SOAP no está soportado directamente por WebSphere ESB y Process Server. Debe configurar una exportación personalizada.

Acerca de esta tarea

WebSphere Integration Developer proporciona una transformación de datos SOAP que puede enlazar a la exportación JMS de WebSphere MQ para crear una exportación SOAP JMS de WebSphere MQ personalizada.

Siga las instrucciones para crear una exportación personalizada para recibir solicitudes SOAP a través del transporte WebSphere MQ para SOAP.

Procedimiento

1. Consulte Visión general de importaciones y exportaciones y Cómo conectarse a WebSphere MQ en la documentación del producto WebSphere Process Server for Multiplatforms V6.2 .
2. Siga la tarea Generación de un enlace de exportación JMS de MQ en la documentación del producto IBM Business Process Manager, Versión 8.6 .

   Utilice el enlace de datos SOAP descrito en Transformaciones de formato de datos JMS empaquetadas previamente para formatear el mensaje SOAP.

Tareas relacionadas

Despliegue de un servicio en Axis 1.4 para utilizarlo en el transporte WebSphere para SOAP utilizando amqwdeployWMQService
Despliegue un servicio Axis 1.4 en el transporte WebSphere MQ para SOAP creando un directorio de despliegue, ejecutando el mandato **amqwdeployWMQService** e iniciando el escucha Axis 1.4 .

Despliegue de un servicio en .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP
Despliegue un servicio .NET Framework 1 o 2 en el transporte WebSphere MQ para SOAP. Cree un directorio de despliegue, ejecute el mandato **amqwdeployWMQService** e inicie el escucha .NET.

Despliegue de un servicio en CICS Transaction Server para utilizar WebSphere Transport para SOAP
El transporte WebSphere MQ para SOAP está integrado en el soporte de servicios web de CICS Transaction Server 4.1 .

Despliegue de un servicio en WebSphere Application Server para utilizar WebSphere Transport para SOAP
WebSphere MQ Transport para SOAP se integra en el bus de integración de servicios en WebSphere Application Server.

Configuración de WebSphere Application Server para utilizar W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 1 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo

SOAP sobre JMS W3C . Configure los recursos de WebSphere MQ y WebSphere Application Server para desarrollar y desplegar el servicio web enlazado a W3C SOAP sobre JMS como transporte.

Despliegue de clientes de servicio web para utilizar el transporte WebSphere MQ para SOAP

Despliegue un cliente de servicio web en uno de varios entornos de cliente diferentes y conéctese a un servicio utilizando el transporte WebSphere MQ para SOAP.

Antes de empezar

Desarrolle el servicio web y despliéguelo para utilizar el transporte WebSphere MQ para SOAP.

Acerca de esta tarea

Puede desplegar un cliente de servicio web para que se ejecute con el transporte WebSphere MQ para SOAP en varios entornos de cliente diferentes. Puede desplegar un cliente Java en Axis 1.4 utilizando solo el software instalado con WebSphere MQ. Para los otros entornos de cliente, debe instalar software adicional.

No está restringido a ejecutar el transporte WebSphere para SOAP en los entornos de cliente para los que hay instrucciones de despliegue. Utilice las instrucciones para desplegar un cliente en uno de los entornos soportados.

Nota: Algunos entornos integrados ofrecen SOAP sobre JMS utilizando el enlace SOAP JMS recomendado W3C , así como el transporte WebSphere MQ para el enlace SOAP. Los releases de WebSphere MQ, hasta 7.0.1.2inclusive, sólo dan soporte al transporte WebSphere MQ para el enlace SOAP. A partir de 7.0.1.3 , puede desplegar clientes Axis2 utilizando un URI que se ajuste a la recomendación de candidato W3C para SOAP sobre JMS. Consulte la guía de aprendizaje Desarrollo de una aplicación de servicios web JAX-WS SOAP/JMS con WebSphere Application Server V7 y Rational Application Developer V7.5.

Despliegue de un cliente de servicio web en Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP

Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione los proxies de cliente y la clase de cliente y configure CLASSPATH. Configure colas y canales de IBM WebSphere MQ , inicie el servicio y pruebe el cliente.

Antes de empezar

Consejo: Despliegue el servicio en HTTP, desarrolle y pruebe el cliente para HTTP y, a continuación, modifique el cliente para el transporte de IBM WebSphere MQ para SOAP:

1. Añada la llamada Register.extension() al cliente.
2. Cambie la dirección de servicio web estática en la clase de localizador de proxy de cliente para utilizar el URI para el transporte IBM WebSphere MQ para SOAP.

Acerca de esta tarea

El despliegue de un cliente Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP requiere un paso de despliegue adicional en comparación con un cliente HTTP. Debe crear un descriptor de despliegue de cliente, `client-config.wsdd`, para correlacionar el transporte `jms:` con la clase de remitente `com.ibm.mq.soap.transport.jms.WMQSender`.

Si utiliza el mandato **amqwdeployWMQService** para generar proxies de cliente, puede desplegar el cliente utilizando los directorios que genera el mandato.

Procedimiento

1. Cree un directorio *deployDir* para que contenga los archivos de despliegue del cliente.

2. Abra una ventana de mandatos en sistemas Windows , o un shell de mandatos utilizando X Window System en sistemas UNIX and Linux , en *deployDir*.
3. Ejecute el mandato **amqwsetcp.cmd** para establecer la CLASSPATH
4. Ejecute el mandato **amqwclientconfig.cmd** para crear un descriptor de despliegue de cliente Axis 1.4 , `client-config.wsdd` en *deployDir*.
5. Asegúrese de que las clases del paquete de cliente, las clases de proxy de cliente y las bibliotecas que utiliza el cliente están en la CLASSPATH.

   amqwdeployWMQService coloca los proxies de cliente .NET en `./generated/server/soap/client/remote/dotnetService` y los proxies Axis 1.4 en `./generated/server/soap/client/remote/`*`client package`*.

Ejemplo

El ejemplo muestra la configuración y salida, Figura 199 en la página 1033, de un cliente Axis 1.4 Java . El cliente, Figura 198 en la página 1032, llama a un servicio web que repite su parámetro de entrada. La definición de servicio, Figura 197 en la página 1032, muestra el URI tomado del WSDL de servicio.

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                        name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

Figura 197. Definición de servicio

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                   + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Figura 198. Cliente de Eje 1.4 Java

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

Figura 199. Configuración y salida de cliente

Qué hacer a continuación

1. Si está desplegando el cliente como un cliente de IBM WebSphere MQ , configure el canal de conexión de cliente y servidor.
2. Si está desplegando el cliente en un gestor de colas diferente al servicio, debe hacer que la cola de destino esté disponible para el cliente. Configure la cola de destino en el gestor de colas de servicio como una cola de clúster, o en el gestor de colas de cliente como una definición de cola remota.

Tareas relacionadas

Despliegue de un cliente de servicio web en Axis2 para utilizar el transporte WebSphere MQ para SOAP
Prepare un directorio de despliegue y un archivo de configuración Axis2 para el cliente. Proporcione los proxies de cliente y la clase de cliente, y configure la variable CLASSPATH. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 4 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Modifique el URL en el cliente Axis2 desarrollado para el transporte WebSphere MQ para SOAP para utilizar la recomendación candidata W3C para SOAP sobre JMS.

Despliegue de un cliente de servicio web en .NET Framework 1 y 2 para utilizar el transporte WebSphere MQ para SOAP
Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione el proxy de cliente y la clase de cliente. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue de un cliente de servicio web en Axis2 para utilizar el transporte WebSphere MQ para SOAP

Prepare un directorio de despliegue y un archivo de configuración Axis2 para el cliente. Proporcione los proxies de cliente y la clase de cliente, y configure la variable CLASSPATH. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Antes de empezar

Consejo: Despliegue el servicio en HTTP. Desarrolle y pruebe el cliente para HTTP y, a continuación, modifique el URL para hacer referencia al servicio utilizando el transporte WebSphere MQ para SOAP.

La tarea muestra cómo desplegar un cliente Axis2 no gestionado en Java Standard Edition. Es posible que desee desplegar un cliente Axis2 en un contenedor web. En “Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse” en la página 1001, ha desarrollado un cliente en un contenedor web y lo ha desplegado en WebSphere Application Server Community Edition. Como parte de la configuración del servidor, ha habilitado la faceta Axis2 e incluido la faceta en la configuración del contenedor web. Para configurar contenedores web en otros servidores de aplicaciones, consulte la

documentación de Axis2 , http://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_container, o la documentación proporcionada con el servidor web.

Nota: Axis2 utiliza el término, contenedor de servlet. Un contenedor de servlet es el mismo que un contenedor web.

Acerca de esta tarea

El despliegue de un cliente Axis2 para utilizar el transporte WebSphere MQ para SOAP es como desplegar un cliente Axis2 para utilizar HTTP. Se necesitan pasos adicionales para proporcionar una vía de acceso de clases a los archivos JAR de WebSphere MQ y para modificar el archivo de configuración Axis2 . El archivo de configuración Axis2 requiere una entrada adicional para JMS. La entrada hace referencia al transporte WebSphere MQ para el archivo JAR SOAP que implementa el `transportSender`de JMS.

Axis2 proporciona un script, `axis2.bat` o `axis2.sh`, que simplifica el despliegue del cliente; consulte los ejemplos de Figura 203 en la página 1036 y Figura 204 en la página 1036.

Nota:

1. `axis2.bat` tiene un error que debe corregirse. La serie `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` debe cambiarse por `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`.
2. En `axis2.bat` y `axis2.sh`, `-Djava.ext.dirs` se utiliza como una forma rápida de hacer referencia a todos los archivos JAR Axis2 , en lugar de añadirlos por separado a la vía de acceso de clases. Desafortunadamente este enfoque es defectuoso, y sólo funciona con algunos JREs. No funciona con los JRE de IBM .

   El parámetro JVM, `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`, hace que los archivos JAR de Axis estén disponibles para la JVM. La JVM intenta crear una instancia de algunos de los archivos JAR de Axis y conduce a un error, cuyos detalles dependen de la JVM. Normalmente, puede ver una de las líneas siguientes en el rastreo de pila:

   ```
   org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
   ```

   o `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`

   La forma correcta de ejecutar un cliente Axis2 no gestionado es añadir los archivos JAR Axis2 a la vía de acceso de clases. La vía de acceso de clases sólo está disponible para la aplicación cliente y no para la JVM.

El procedimiento describe los pasos generales para ejecutar un cliente Axis2 no gestionado sin utilizar el script axis2 . Los ejemplos de Figura 201 en la página 1035 y Figura 202 en la página 1036 son scripts para Windows y Linux.

Procedimiento

1. Descargue Axis2 1.4.1 de http://ws.apache.org/axis2/download/1_4_1/download.cgi y desempaquete en una carpeta, `Axis2-1.4.1`.
2. Actualice `axis2.xml` en `Axis2-1.4.1\conf`.
   a) Actualice `axis2.xml` en `Axis2-1.4.1\conf`. Añada el transporte WebSphere MQ para SOAP como `transportSender`:

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) Si es necesario, modifique el tamaño de la agrupación de conexiones del valor predeterminado de 10.

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   <parameter name="ResourcePoolCapacity">20</parameter>
   </transportSender>
   ```

`Capacidad deResourcePool` define cuántas entradas de punto final de servicio se conservan en la memoria caché. El valor debe ser como mínimo 1. Si el número de entradas de punto final de servicio supera el tamaño de memoria caché, las entradas se suprimen para dejar espacio para nuevas entradas. El tamaño de una entrada de punto final varía. Establezca un número que sea lo suficientemente grande para evitar la hiperpaginación de la memoria caché.

Consulte el paso 3 en "Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse" en la página 1001.

3. Cree un directorio *deployDir*. En este directorio, copie la estructura de carpetas que contiene los proxies de cliente y cliente. *deployDir* es equivalente a la carpeta *`project`*`\bin` en un proyecto Java de Eclipse .
4. Abra una ventana de mandatos en Windows, o un shell de mandatos utilizando X Window System en sistemas UNIX and Linux , en *deployDir*.
5. Actualice la vía de acceso de clases para incluir el directorio actual, los archivos JAR de Axis2 , `com.ibm.mqjms.jar` y `com.ibm.mq.axis2.jar`.

   `com.ibm.mqjms.jar` hace referencia a todos los demás archivos JAR de WebSphere MQ necesarios.
6. Utilice el mandato **Java** para iniciar el programa cliente.

Ejemplos

Cuatro ejemplos de ejecución de un cliente Axis2 se listan en Figura 202 en la página 1036 en Figura 204 en la página 1036. Figura 200 en la página 1035 muestra la salida de la ejecución del cliente asíncrono listado en Figura 183 en la página 1006.

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

Figura 200. Salida de la ejecución de SQA2AsyncClient

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

Figura 201. runpojo.bat: Windows, utilizando una vía de acceso de clases

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

Figura 202. runpojo.sh: Linux, utilizando una vía de acceso de clases.

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

Figura 203. runaxis2.bat: Windows, utilizando axis2.bat

Nota

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

Figura 204. runaxis2.sh: Linux, utilizando axis2.sh

Nota

Tareas relacionadas

Despliegue de un cliente de servicio web en Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP

Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione los proxies de cliente y la clase de cliente y configure CLASSPATH. Configure colas y canales de IBM WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS

Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 4 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Modifique el URL en el cliente Axis2 desarrollado para el transporte WebSphere MQ para SOAP para utilizar la recomendación candidata W3C para SOAP sobre JMS.

Despliegue de un cliente de servicio web en .NET Framework 1 y 2 para utilizar el transporte WebSphere MQ para SOAP

Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione el proxy de cliente y la clase de cliente. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS

Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 4 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Modifique el URL en el cliente Axis2 desarrollado para el transporte WebSphere MQ para SOAP para utilizar la recomendación candidata W3C para SOAP sobre JMS.

Antes de empezar

Primero debe completar la tarea, "Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse" en la página 1001 para llamar a **SimpleJavaListener** utilizando un cliente Axis2 y el transporte WebSphere MQ para el protocolo SOAP.

También debe haber creado el servicio web y configurado WebSphere MQ y WebSphere Application Server en las tareas anteriores:

1. "Configurar recursos de WebSphere MQ" en la página 1026.
2. "Configurar recursos de WebSphere Application Server" en la página 1027.
3. "Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS" en la página 987.

En la tarea, el cliente se ejecuta en Eclipse Galileo. Puede ejecutar el cliente desde la línea de mandatos modificando el archivo `Axis2.bat` que se suministra con Axis2.

Acerca de esta tarea

El único cambio que debe realizar en el cliente estático de Axis2 `StockQuoteAxis` existente para llamar al servicio Axis StockQuotealojado en WebSphere Application Server es cambiar el URL pasado al cliente. Puesto que el WSDL no ha cambiado, puede utilizar las mismas clases de proxy en el paquete `soap.server`.

Tiene dos enfoques para definir el URL que se debe pasar al cliente. Puede utilizar el mismo URL que en el `StockQuoteAxis.wsdl`generado. Debe añadir los parámetros `jndiInitialContextFactory` y `jndi`URL para acceder al directorio JNDI de WebSphere Application Server. Otro enfoque es cambiar el URL y proporcionar al cliente acceso directo a las colas `REQUESTAXIS` y `REPLYAXIS` en `QM1`, sin utilizar una búsqueda JNDI.

Los parámetros de conexión que defina en el URL pasado al cliente Axis2 se utilizan para conectarse al gestor de colas de WebSphere MQ y a las colas necesarias para enviar y recibir mensajes SOAP. El servicio no utiliza necesariamente los parámetros de conexión pasados al cliente Axis2 . Puede utilizar las prestaciones de gestión de colas distribuidas de WebSphere MQ para desacoplar el cliente y el servicio de la utilización del mismo gestor de colas o del mismo servidor de nombres.

Procedimiento

1. Guarde el URL del `StockQuoteAxis.wsdl` generado y cierre Rational Application Developer para ahorrar en memoria.

   Si no ha cambiado la configuración del servidor, al cerrar Rational Application Developer se detiene el servidor de aplicaciones. En este caso, inicie el servidor con el mandato:

   ```
   startserver server1
   ```

2. Abra Eclipse Galileo en el espacio de trabajo con el proyecto de cliente Axis2 .
3. Abra `SQA2StaticClient.java`.

   Consulte SQA2StaticClient.java.
4. Llame al servicio utilizando la variante `queue` del URI.
   a) Modifique el URL.

   El nuevo URI es:

   ```
   jms:queue:REQUESTAXIS
             ?replyToName=REPLYAXIS
             &connectionFactory=connectQueueManager(QM1)Bind(Server)
             &targetService=StockQuoteAxis;
   ```

   Compare esto con el URL de `StockQuoteAxis.wsdl`:

```
jms:jndi:requestaxis
         ?jndiConnectionFactoryName=qm1
         &targetService=StockQuoteAxis
```

Figura 205. URL de `StockQuoteAxis.wsdl`

- REQUESTAXIS está ahora en mayúsculas ya que es un nombre de cola y no un nombre JNDI.
- La conexión con QM1 es sencilla.
- El URI no contiene el nombre del destino de respuesta. El cliente debe definir la cola en la que espera respuestas.

b) Ejecute `SQA2StaticClient.java` utilizando el mismo **Ejecutar como ...** configuración como ha hecho en la tarea, "Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse" en la página 1001.

5. Llame al servicio utilizando la variante `jndi` del URI, utilizando WebSphere Application Server como servidor de nombres.

a) Utilice el URL de `StockQuoteAxis.wsdl`, Figura 205 en la página 1038, proporcionando los parámetros que faltan para utilizar el servicio de denominación en WebSphere Application Server.

Los parámetros y valores que faltan que debe proporcionar son:

Tabla 151. Parámetros JNDI adicionales

Parámetro	Valor utilizado en este ejemplo	Descripción
&jndiURL	`iiop://localhost:2810` o `corbaname:iiop:localhost:2810`	URI del proveedor de nombres. Para WebSphere Application Server, el valor predeterminado es 2809. También se conoce como el número de puerto del conector RMI y el puerto de rutina de carga. El valor se lista en `SystemOut.log` `00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810`
&jndiInitialContextFactory	`com.ibm.websphere.naming.WsnInitialContextFactory`	El nombre de la fábrica de contexto inicial utilizada por WebSphere Application Server.
&replyToName	`replyaxis`	Nombre JNDI de la cola REPLYAXIS .

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) Añada los archivos JAR necesarios para la búsqueda JNDI.

En esta configuración, se han añadido los siguientes archivos JAR a la vía de acceso de compilación para ejecutar la tarea utilizando la variante `jndi` del URL JMS:

- `com.ibm.jaxws.thinclient_7.0.0.jar` de *`Rational install directory`*`\SDP\runtimes\base_v7\runtimes`.

- `com.ibm.ws.runtime.jar` desde *`Rational install directory`*`\SDP\runtimes\base_v7\plugins`

Para un proveedor JNDI diferente, necesita archivos JAR diferentes.

Los otros archivos JAR de la vía de acceso de construcción son:

i) Todos los archivos JAR en *`WebSphere MQ Install directory`*`\java\lib`.

ii) Todos los archivos JAR en *`Axis2-1.5.1`*`\lib`.

iii) Java 6.0 JRE.

c) Ejecute `SQA2StaticClient.java` utilizando el mismo **Ejecutar como ...** configuración como ha hecho en la tarea, "Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse" en la página 1001.

Resultados

En ambos casos, la respuesta del servicio se visualiza en la vista de consola de cliente.

Tareas relacionadas

Despliegue de un cliente de servicio web en Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP
Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione los proxies de cliente y la clase de cliente y configure CLASSPATH. Configure colas y canales de IBM WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue de un cliente de servicio web en Axis2 para utilizar el transporte WebSphere MQ para SOAP
Prepare un directorio de despliegue y un archivo de configuración Axis2 para el cliente. Proporcione los proxies de cliente y la clase de cliente, y configure la variable CLASSPATH. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue de un cliente de servicio web en .NET Framework 1 y 2 para utilizar el transporte WebSphere MQ para SOAP
Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione el proxy de cliente y la clase de cliente. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue de un cliente de servicio web en .NET Framework 1 y 2 para utilizar el transporte WebSphere MQ para SOAP

Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione el proxy de cliente y la clase de cliente. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Antes de empezar

Consejo: Desarrolle y pruebe el servicio y el cliente utilizando Visual Studio. A continuación, modifique el cliente para el transporte WebSphere MQ para SOAP.

1. Si está desplegando un servicio utilizando .NET Framework 1 o 2, cree el servicio como una biblioteca (`.dll`). Despliegue utilizando el transporte WebSphere MQ para SOAP.
2. Añada la llamada Register.Extension() al cliente.
3. Añada una referencia a amqsoap.dll, que se encuentra en *`MQ_Install`*`\bin`.
4. Cambie la propiedad `Url` estática en el constructor de clase de proxy de cliente por el URI `jms:/` , para el transporte WebSphere MQ para SOAP.

Acerca de esta tarea

El despliegue de un cliente de servicio web para .NET Framework 1 o 2 para utilizar el transporte WebSphere MQ para SOAP requiere un paso de despliegue adicional. Debe registrar `amqsoap.dll`

con .NET Framework. `amqsoap.dll` se registra automáticamente como parte de la instalación de WebSphere MQ Transport para SOAP, pero es posible que tenga que registrarlo de nuevo.

Si utiliza el mandato **amqwdeployWMQService** para generar proxies de cliente, puede desplegar el cliente utilizando los directorios que genera el mandato.

Procedimiento

1. Cree un directorio *deployDir* para que contenga los archivos de despliegue del cliente.
2. Abra una ventana de mandatos en *deployDir*.
3. Ejecute **amqwsetcp** para establecer CLASSPATH si el servicio se va a ejecutar en Axis 1.4.
4. Si es necesario, ejecute **amqwRegisterDotNet** para registrar `amqsoap.dll` con .NET Framework.

Ejemplo

El ejemplo muestra la configuración y salida, Figura 208 en la página 1040, de un cliente V2 de .NET Framework. El cliente, Figura 207 en la página 1040, llama a un servicio web que repite su parámetro de entrada. La definición de URL estático, Figura 206 en la página 1040, muestra el constructor para el proxy de cliente.

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
               &connectionFactory=(connectQueueManager(QM1)binding(server))
               &initialContextFactory=com.ibm.mq.jms.Nojndi
               &targetService=Quote.asmx
               &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

Figura 206. Constructor de proxy de cliente estático

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

Figura 207. Programa cliente

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

Figura 208. Configuración y salida

Qué hacer a continuación

1. Si está desplegando el cliente como un cliente MQI de WebSphere MQ , configure el cliente y el canal de conexión de servidor.

2. Si está desplegando el cliente en un gestor de colas diferente al servicio, debe hacer que la cola de destino esté disponible para el cliente. Configure la cola de destino en el gestor de colas de servicio como una cola de clúster, o en el gestor de colas de cliente como una definición de cola remota.

Tareas relacionadas

Despliegue de un cliente de servicio web en Axis 1.4 para utilizar el transporte de IBM WebSphere MQ para SOAP
Prepare un directorio de despliegue y un descriptor de despliegue para el cliente. Proporcione los proxies de cliente y la clase de cliente y configure CLASSPATH. Configure colas y canales de IBM WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue de un cliente de servicio web en Axis2 para utilizar el transporte WebSphere MQ para SOAP
Prepare un directorio de despliegue y un archivo de configuración Axis2 para el cliente. Proporcione los proxies de cliente y la clase de cliente, y configure la variable CLASSPATH. Configure colas y canales de WebSphere MQ , inicie el servicio y pruebe el cliente.

Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS
Un servicio web enlazado a la recomendación candidata W3C para SOAP sobre JMS debe ejecutarse en el contenedor EJB de un servidor de aplicaciones Java EE . Esta tarea es el paso 4 de conectar un cliente de servicio web Axis2 y un servicio web desplegado en WebSphere Application Server utilizando el protocolo SOAP sobre JMS W3C . Modifique el URL en el cliente Axis2 desarrollado para el transporte WebSphere MQ para SOAP para utilizar la recomendación candidata W3C para SOAP sobre JMS.

Conecte un cliente Axis2 a un servicio JAX-WS utilizando W3C SOAP sobre JMS y WebSphere Application Server

Cuando complete esta tarea, habrá llamado a un servicio web JAX-WS que se ejecuta en WebSphere Application Server desde un cliente Axis2 . El cliente Axis2 y WebSphere Application Server utilizan la recomendación candidata W3C para el protocolo SOAP sobre JMS que se ejecuta en WebSphere MQ. Utilice Eclipse Galileo y Rational Application Developer para crear el cliente de servicio web y el servicio web, respectivamente.

Antes de empezar

La tarea requiere la versión 7 de Rational Software Development Environment y WebSphere Application Server. La tarea se ha creado utilizando Rational Application Developer empaquetado con Rational Software Architect for WebSphere Software v7.5.5.1y WebSphere Application Server Versión 7.0 Test Environment v7.0.0.9 Actualización 1. También necesita WebSphere MQ v7.0.1.3.

La tarea se basa en otras dos tareas, “Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse” en la página 980y “Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse” en la página 1001. Para completar estas tareas, el entorno de desarrollo ya tiene instalado Eclipse Galileo, WASCE, el plugin Eclipse para WASCE y Axis2 1.4.1 . No necesita WASCE para esta tarea.

Algunos de los pasos son complejos. Los pasos asumen cierta familiaridad con el desarrollo de aplicaciones de servicio web para WebSphere Application Server utilizando Rational Application Developer. Las demandas de procesador y memoria de la tarea son grandes. La tarea se ha realizado en una máquina virtual VMWare Windows XP SP3 asignada 1.8GB de memoria.

Instale todo el software antes de iniciar la tarea. El software tarda aproximadamente un día en descargarse y un día en instalarse, en función del ancho de banda. La tarea tarda al menos medio día.

Acerca de esta tarea

El escenario de esta tarea es que ha desarrollado un servicio web de cotización bursátil, `StockQuoteAxis`, utilizando una herramienta de código abierto, Eclipse Galileo. `StockQuoteAxis` se despliega utilizando SOAP sobre HTTP que se ejecuta en un servidor de código abierto, WASCE.

Desea enlazar los servicios web que despliega a un transporte de mensajería basado en estándares, como SOAP sobre JMS, o a mensajería fiable de servicios web, así como a SOAP sobre HTTP. Desea

que tanto el cliente como el servicio utilicen interfaces basadas en estándares. Por este motivo, aunque el equipo de desarrollo de proyectos futuros ha implementado una solución utilizando el transporte WebSphere MQ para SOAP, no ha entrado en producción.

El cliente Axis2 ha eliminado el problema de que el cliente SOAP para el transporte WebSphere MQ para SOAP requería un cambio del cliente HTTP. El problema sigue siendo que el servicio conectado mediante el transporte IBM WebSphere MQ para SOAP está alojado en un escucha especial proporcionado por WebSphere MQ: `SimpleJavaListener`.

Con el estándar SOAP sobre JMS W3C en estado de recomendación de candidato, algunos proveedores proporcionan soporte para SOAP sobre JMS W3C . El soporte le permite desplegar un servicio web en un servidor de aplicaciones y conectarse al mismo servicio utilizando diversos protocolos de conectividad. El soporte proporcionado por WebSphere Application Server v7 elimina el problema de tener que alojar el servicio web por separado para utilizar un transporte SOAP basado en mensajes. El uso de una interfaz de transporte de mensajes basada en estándares, JMS, significa que puede desarrollar soluciones utilizando herramientas de distintos proveedores. Espera que las herramientas de servicios web en Eclipse incluyan enlaces SOAP sobre JMS en el futuro.

La mayoría de los pasos se realizan utilizando Eclipse, o las herramientas de gestión que se proporcionan con los productos WebSphere . Los pasos se describen para un entorno Windows . Con ligeras modificaciones en algunos mandatos, puede realizar los pasos en otras plataformas.

Se listan los pasos preliminares para crear el servicio web HTTP y conectarse a él utilizando Axis2 . El cliente, y WSDL, de estos pasos se utilizan para crear la solución

Procedimiento

1. Conectar con el servicio web Axis StockQuoteutilizando un cliente Axis2 y el transporte IBM WebSphere MQ para SOAP
   a) “Desarrollo de un servicio JAX-RPC para el transporte WebSphere MQ para SOAP utilizando Eclipse” en la página 980
   b) “Desarrollo de un cliente JAX-WS para WebSphere Transport para SOAP utilizando Eclipse” en la página 1001
   c) “Despliegue de un cliente de servicio web en Axis2 para utilizar el transporte WebSphere MQ para SOAP” en la página 1033
2. Conéctese al servicio web de eje StockQuoteutilizando un cliente Axis2 y la recomendación de candidato W3C para SOAP sobre JMS.
   a) “Configurar recursos de WebSphere MQ” en la página 1026
   b) “Configurar recursos de WebSphere Application Server” en la página 1027
   c) “Desarrollo de un servicio web EJB JAX-WS para W3C SOAP sobre JMS” en la página 987
   d) “Despliegue en un cliente Axis2 utilizando W3C SOAP sobre JMS” en la página 1036

Puente WebSphere MQ para HTTP

Con el puente WebSphere MQ para HTTP, las aplicaciones cliente pueden intercambiar mensajes con WebSphere MQ sin necesidad de instalar un cliente MQI de WebSphere MQ . Puede llamar a WebSphere MQ desde cualquier plataforma o idioma con prestaciones HTTP.

Introducción al puente WebSphere MQ para HTTP

El puente WebSphere MQ para HTTP es una aplicación web Java, Enterprise Environment (JEE). Los clientes HTTP pueden enviarle solicitudes **POST**, **GET**y **DELETE** para colocar, examinar y suprimir mensajes de las colas de WebSphere MQ . El puente WebSphere MQ para HTTP no es adecuado para su uso con mensajes, si es necesaria una entrega asegurada.

Ventajas

Con el puente WebSphere MQ para HTTP, puede enviar y recibir mensajes de WebSphere MQ utilizando HTTP desde una amplia variedad de entornos:

- Entornos que dan soporte a HTTP, pero no a WebSphere MQ.
- Entornos que no tienen suficiente espacio de almacenamiento para instalar un cliente MQI de WebSphere MQ .
- Entornos que son demasiado numerosos para instalar el cliente MQI de WebSphere MQ en cada sistema que requiere acceso a WebSphere MQ.
- Aplicaciones basadas en web desde las que desea enviar o recibir mensajes sin codificar su propio puente con WebSphere MQ.
- Aplicaciones basadas en web que desea mejorar, utilizando técnicas asíncronas como AJAX. WebSphere MQ para HTTP hace que las colas y temas de WebSphere MQ estén disponibles utilizando REST (Representation State Transfer) sobre HTTP.

El soporte HTTP se puede utilizar con topologías de mensajería punto a punto y de publicación/suscripción.

¿Cómo funciona el soporte HTTP?

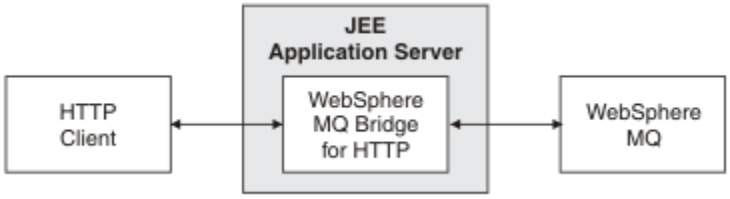


Figura 209. Puente WebSphere MQ para HTTP

El puente WebSphere MQ para la aplicación web HTTP recibe solicitudes HTTP de uno o varios clientes. Interactúa con WebSphere MQ en su nombre y les devuelve respuestas HTTP.

El puente WebSphere MQ para HTTP es un servlet JEE que está conectado a WebSphere MQ utilizando un adaptador de recursos. El servlet HTTP maneja tres tipos diferentes de solicitudes HTTP: **POST**, **GET**y **DELETE**.

Tabla 152. WebSphere MQ Bridge for HTTP Verbs

Solicitud HTTP	**Resultado**
POST	Coloca un mensaje en una cola o tema.
GET	Examina el primer mensaje de una cola. En línea con el protocolo HTTP, **GET** no suprime el mensaje de la cola. No utilice **GET** con mensajería de publicación/suscripción.
Suprimir	Obtiene y suprime un mensaje de una cola o tema.

Ejemplo de HTTP POST

HTTP **POST** pone un mensaje en una cola o una publicación en un tema. El ejemplo Java **HTTPPOST** es un ejemplo de petición HTTP **POST** de un mensaje a una cola. También se puede crear una petición HTTP **POST** utilizando un formulario de navegador o un kit de herramientas AJAX en lugar de Java.

La Figura 210 en la página 1044 muestra una solicitud HTTP para colocar un mensaje en una cola denominada `myQueue`. Esta solicitud contiene la cabecera HTTP `x-msg-correlId` para establecer el ID de correlación del mensaje WebSphere MQ .

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

*Figura 210. Ejemplo de una solicitud HTTP **POST** a una cola*

Figura 211 en la página 1044 muestra la respuesta devuelta al cliente. No hay contenido de respuesta.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Figura 211. Ejemplo de una respuesta de HTTP POST

Ejemplo de HTTP DELETE

HTTP **DELETE** obtiene un mensaje de una cola y suprime el mensaje o recupera y suprime una publicación. El ejemplo Java **HTTPDELETE** es un ejemplo de petición HTTP **DELETE** que lee un mensaje de una cola. También se puede crear una petición HTTP **DELETE** utilizando un formulario de navegador o un kit de herramientas AJAX en lugar de Java.

Figura 212 en la página 1044 es una solicitud HTTP para suprimir el siguiente mensaje en la cola denominada `myQueue`. En respuesta, se devuelve el cuerpo del mensaje al cliente. En términos de WebSphere MQ , HTTP **DELETE** es una obtención destructiva.

La solicitud contiene la cabecera de solicitud HTTP `x-msg-wait`, que indica al puente de WebSphere MQ para HTTP cuánto tiempo debe esperar a que llegue un mensaje a la cola. La solicitud también contiene la cabecera de solicitud `x-msg-require-headers`, que especifica que el cliente debe recibir el ID de correlación de mensaje en la respuesta.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figura 212. Ejemplo de una solicitud HTTP **DELETE***

Figura 213 en la página 1044, es la respuesta devuelta al cliente. Se devuelve el ID de correlación al cliente, tal como se ha solicitado en `x-msg-require-headers` de la solicitud.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

*Figura 213. Ejemplo de una respuesta de HTTP **DELETE***

Ejemplo de HTTP GET

HTTP **GET** obtiene un mensaje de una cola. El mensaje permanece en la cola. En términos de WebSphere MQ , HTTP **GET** es una solicitud de examen. Una petición HTTP **GET** se puede crear utilizando un cliente Java, un formulario de navegador o un kit de herramientas AJAX.

Figura 214 en la página 1045 es una solicitud HTTP para examinar el siguiente mensaje en la cola denominada `myQueue`.

La solicitud contiene la cabecera de solicitud HTTP `x-msg-wait`, que indica al puente de WebSphere MQ para HTTP cuánto tiempo debe esperar a que llegue un mensaje a la cola. La solicitud también contiene la cabecera de solicitud `x-msg-require-headers`, que especifica que el cliente debe recibir el ID de correlación de mensaje en la respuesta.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figura 214. Ejemplo de una solicitud HTTP **GET***

Figura 215 en la página 1045 es la respuesta devuelta al cliente. Se devuelve el ID de correlación al cliente, tal como se ha solicitado en `x-msg-require-headers` de la solicitud.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

*Figura 215. Ejemplo de una respuesta de HTTP **GET***

Instalación, configuración y verificación del puente WebSphere MQ para HTTP

Obtenga el puente WebSphere MQ para HTTP instalando "Java Messaging and Web Services" desde los materiales de instalación de cliente o servidor MQI de WebSphere MQ . Despliegue el puente WebSphere MQ para HTTP en un servidor de aplicaciones adecuado.

Antes de empezar

Compruebe los productos que son requisito previo en Requisitos del sistema para IBM WebSphere MQ. El proceso de instalación no comprueba la presencia y disponibilidad del software de requisito previo para ejecutar el puente WebSphere MQ para HTTP. Hay que verificar que los requisitos previos están instalados.

El puente WebSphere MQ para HTTP se ejecuta en cualquier servidor de aplicaciones compatible con Java EE 1.4 , instalando el adaptador de recursos WebSphere MQ . También puede ejecutar el puente WebSphere MQ para HTTP en un release de WebSphere Application Server anterior a la versión 6.0.2.1. Utilice WebSphere Application Server Message Listener Port (MLP) para integrar WebSphere MQ como proveedor JMS.

El soporte para el puente WebSphere MQ para HTTP solo se proporciona para los siguientes servidores de aplicaciones:

- WebSphere Application Server 6.0.2.1 y posterior.
- WebSphere Application Server Community Edition Versión 1.1 y posterior.

Acerca de esta tarea

El puente WebSphere MQ para HTTP se proporciona como un archivo `.war` , `WMQHTTP.war`.

- En plataformas UNIX y Linux,
  - `WMQHTTP.war` se incluye como parte de la opción de instalación "Java Messaging and Web Services" . La opción está disponible en los materiales de instalación del cliente y del servidor.
  - `WMQHTTP.war` se instala en *<mqmtop>*`/java/http/WMQHTTP.war`. *< mqmtop>* es el directorio donde está instalado WebSphere MQ .
  - `WMQHTTP.samples` se instala en *<mqmtop>*`/java/http/samples`. *< mqmtop>* es el directorio donde está instalado WebSphere MQ .

Lleve a cabo los siguientes pasos de instalación para instalar el puente WebSphere MQ para HTTP, despliéguelo y configúrelo, y verifique la configuración. Los detalles de los pasos de configuración varían en distintos servidores de aplicaciones. Utilice "Despliegue y verificación del puente WebSphere MQ para HTTP en WebSphere Application Server V6.1.0.9" en la página 1047 como plantilla para los pasos a seguir en el servidor de aplicaciones.

Procedimiento

1. Obtenga `WMQHTTP.war` instalando el cliente o servidor MQI de WebSphere MQ .
2. Copie `WMQHTTP.war` en un servidor desde el que se pueda desplegar en un servidor de aplicaciones.
3. Despliegue `WMQHTTP.war` en un servidor de aplicaciones.
4. Si es necesario, instale WebSphere MQ como adaptador de recursos en el servidor de aplicaciones.

   Averigüe si WebSphere MQ ya está configurado como proveedor de mensajería en el servidor de aplicaciones. Utilice la herramienta de administración o gestión que se proporciona con el servidor de aplicaciones para buscar WebSphere MQ. WebSphere MQ se puede encontrar en la vía de acceso siguiente, **Recursos > JMS > Proveedores de mensajería**.
5. Configure una fábrica de conexiones en el servidor de aplicaciones para conectarse a un gestor de colas que utilice el transporte de cliente MQI de WebSphere MQ[12].
6. Configure la aplicación `WMQHTTP.war` Web en el servidor de aplicaciones para utilizar la fábrica de conexiones
7. Compruebe la configuración.
   a) Configure el gestor de colas especificado en la fábrica de conexiones y una cola local.
   b) Coloque un mensaje en la cola local.
   c) Cree el canal de conexión de servidor especificado en la fábrica de conexiones, con autorización para leer y escribir en la cola local.
   d) Inicie el gestor de colas y el escucha.
   e) Inicie el servidor de aplicaciones y `WMQHTTP.war`, si todavía no se están ejecutando.
   f) Abra un navegador y escriba `http://`*`hostname`*`:`*`web port`*`/`*`Context root`*`/msg/queue/`*`local queue`*

Resultados

La ventana del navegador muestra el mensaje que ha colocado en la cola local.

Qué hacer a continuación

1. Pruebe el ejemplo, "Despliegue y verificación del puente WebSphere MQ para HTTP en WebSphere Application Server V6.1.0.9" en la página 1047.
2. Ejecute las aplicaciones Java HTTP de ejemplo.

[12] Inicialmente, al menos, configure el transporte de cliente. Algunos servidores de aplicaciones pueden conectarse a WebSphere MQ utilizando conexiones directas o en modalidad de enlaces.

Despliegue y verificación del puente WebSphere MQ para HTTP en WebSphere Application Server V6.1.0.9

Utilice el ejemplo siguiente para preparar un despliegue del puente WebSphere MQ para HTTP para ejecutar los programas Java HTTP de ejemplo. El despliegue se realiza en WebSphere Application Server V6.1.0.9.

Antes de empezar

1. Siga las instrucciones de "Instalación, configuración y verificación del puente WebSphere MQ para HTTP" en la página 1045, para copiar `WMQHTTP.war` en un servidor accesible para la instalación de WebSphere Application Server.
2. Configure un gestor de colas, y una cola, que se utilizará para probar la configuración:
   - En el ejemplo, el gestor de colas se configura como que utiliza los valores de Tabla 153 en la página 1047:

   Tabla 153. Configuración del gestor de colas

Objeto	**Valor**
Nombre de host	`itso-01`
Gestor de colas	`QM1`
Cola local	`HTTPTESTQ`
Canal de conexión de servidor	`MYSVRCON`. Configure un ID de usuario de MCA con autorización suficiente para leer y escribir en `HTTPTESTQ`.
Puerto de escucha	`1414`

3. Iniciar el gestor de colas y el escucha
4. Coloque un mensaje de prueba en `HTTPTESTQ`. Por ejemplo:
   a. Inicie WebSphere MQ Explorer.
   b. En la lista de colas locales para QM1, pulse con el botón derecho **HTTPTESTQ > Colocar mensaje de prueba > tipo `First Message` > Colocar mensaje > Cerrar**
5. Inicie el servidor de aplicaciones e inicie sesión en Integrated Solutions Console.

Acerca de esta tarea

El ejemplo muestra los pasos a seguir si ejecuta WebSphere Application Server V6.1.0.9 como servidor de aplicaciones. Si está ejecutando una versión diferente de WebSphere Application Server, o está ejecutando un servidor de aplicaciones diferente, los pasos son diferentes. WebSphere Application Server V6.1.0.9 está preconfigurado con WebSphere MQ instalado como proveedor de mensajes, utilizando las bibliotecas de cliente MQI de WebSphere MQ . Si WebSphere MQ no está preconfigurado como proveedor de mensajería, o si desea utilizar enlaces de servidor WebSphere MQ , debe instalar y configurar el adaptador de recursos WebSphere MQ para JEE en el servidor de aplicaciones.

Siga las instrucciones para desplegar WebSphere MQ Bridge for HTTP en WebSphere Application Server V6.1.0.9y verifique el despliegue utilizando un navegador:

Procedimiento

1. En el panel de navegación, pulse **Recursos > Proveedores JMS > Proveedor de mensajería de WebSphere MQ**.

   Puede configurar a nivel de nodo, célula o servidor, en función del despliegue de WebSphere Application Server. El ejemplo utiliza el despliegue a nivel de servidor.
2. En **Propiedades adicionales**, pulse **Fábricas de conexiones > Nueva**.

3. En el formulario de proveedores JMS, proporcione la información en Tabla 154 en la página 1048, o las alternativas que elija, pulse **Aplicar > Guardar**.

Tabla 154. Establecer o modificar los campos siguientes

Campo	**Valor**
Nombre	`WMQHTTPBridge`
Nombre de JNDI	`jms/WMQHTTPJCAConnectionFactory`
Gestor de colas	`QM1`
Host	`itso-01`
Puerto	`1414`
Canal	`MYSVRCON`
Tipo de transporte	`CLIENT`

4. En el panel de navegación, pulse **Aplicaciones > Instalar nueva aplicación**.
5. Inserte la vía de acceso a `WMQHTTP.war` en el formulario y proporcione una raíz de contexto, pulse **Siguiente**.
   a) La raíz de contexto es opcional. `mq` es la raíz de contexto predeterminada para las aplicaciones HTTP de ejemplo.
   b) La raíz de contexto forma parte del URI que identifica el puente WebSphere MQ para HTTP. Puede omitir la raíz de contexto o cambiarla más tarde.
6. En la página **Seleccionar opciones de instalación** del asistente de instalación, no tiene que cambiar ninguno de los valores predeterminados, pulse **Siguiente**.
7. En la página **Correlacionar módulos con servidores** , seleccione un clúster o servidor, seleccione el recuadro Seleccionar y pulse **Aplicar > Siguiente**.
8. En la página **Correlacionar referencias de recursos con recursos** , en el formulario **javax.jms.ConnectionFactory** , pulse **Examinar ...** en el puente IBM WebSphere MQ para la fila HTTP.
9. En la página **Aplicaciones empresariales > Recursos disponibles** , seleccione **WMQHTTPBridge**y pulse **Aplicar**.
10. De nuevo en el formulario **javax.jms.ConnectionFactory** , seleccione el método de autenticación.
    a) Para el ejemplo, elija **Ninguno**, pulse **Aplicar**. Las otras opciones requieren una configuración adicional.
11. Marque el recuadro de selección **Seleccionar** para IBM WebSphere MQ Bridge for HTTP, pulse **Siguiente > Siguiente > Finalizar > Guardar**
12. En el panel de navegación, pulse **Aplicaciones > Aplicaciones de empresa**.
13. Marque el recuadro de selección para `WMQHTTP.war`y pulse **Iniciar**.
14. Abra una ventana de navegador. Escriba `http://itso-01:9080/mq/msg/queue/HTTPTESTQ`, utilizando el nombre de host y el puerto adecuados.

Resultados

La ventana del navegador muestra `First Message`, si la configuración es satisfactoria.

Qué hacer a continuación

Ejecute las aplicaciones Java HTTP de ejemplo.

Publicación/suscripción utilizando el puente WebSphere MQ para HTTP

El puente WebSphere MQ para HTTP utiliza la interfaz de publicación/suscripción de WebSphere MQ classes for JMS. HTTP **POST** crea una publicación. HTTP **DELETE** crea una suscripción gestionada no duradera. Debe configurar la publicación/suscripción para JMS antes de utilizar el URI de tema.

La publicación/suscripción está totalmente integrada en WebSphere MQ en la versión 7. Antes de la versión 7, un intermediario de publicación/suscripción independiente manejaba publicaciones y suscripciones. Se denomina publicación/suscripción "en cola" , para distinguirla de la publicación/suscripción totalmente integrada en la versión 7. La versión 7 emula la publicación/suscripción en cola utilizando la publicación/suscripción integrada. La emulación permite que las aplicaciones de publicación/suscripción en cola existentes coexistan con las aplicaciones integradas que se ejecutan en el mismo gestor de colas. Las aplicaciones de publicación/suscripción en cola también pueden interoperar con aplicaciones integradas, compartiendo los mismos temas. En la versión 6, el intermediario se suministraba con WebSphere MQ; antes de la versión 6 estaba disponible como SupportPack.

Configuración

El puente WebSphere MQ para HTTP utiliza la interfaz JMS para publicar y suscribirse. En la versión 7, puede controlar si las clases WebSphere MQ para JMS utilizan la publicación/suscripción en cola o integrada, utilizando la propiedad JMS `PROVIDERVERSION` .

Una consideración adicional es que puede utilizar bibliotecas de cliente MQI de WebSphere MQ con el puente WebSphere MQ para HTTP o bibliotecas de servidor. Las bibliotecas de cliente de la versión 6 solo dan soporte a la publicación/suscripción en cola, mientras que las bibliotecas de la versión 7 dan soporte a la publicación/suscripción en cola e integrada. La mayoría de servidores web o de aplicaciones que utilizan WebSphere MQ como proveedor de mensajería lo hacen utilizando bibliotecas de cliente. Para poder utilizar la publicación/suscripción integrada, las bibliotecas de cliente y servidor MQI de WebSphere MQ deben estar al menos en la versión 7. Si cualquiera de los dos ejecuta una versión anterior de WebSphere que no sea 7, debe configurar la publicación/suscripción en cola; consulte Tabla 155 en la página 1049. Compruebe qué bibliotecas están instaladas o configuradas con el servidor web o el servidor de aplicaciones que está utilizando.

Tabla 155. Modalidades de configuración de publicación/suscripción

	Cliente V6 o anterior	**Cliente V7 o posterior**
Servidor V6 o anterior	1. Ejecute el script `\java\bin\MQJMS_PSQ.mqsc`	No soportado
Servidor V7 o posterior	1. Ejecute el script `\java\bin\MQJMS_PSQ.mqsc` 2. Establezca el gestor de colas en `PSMODE=ENABLED`	1. Si PROVIDERVERSION = 7 a. Establezca el gestor de colas en `PSMODE=ENABLED` o `PSMODE=COMPAT` 2. Si PROVIDERVERSION = 6 a. Establezca el gestor de colas en `PSMODE=ENABLED`

Publicar

Envíe una solicitud HTTP **POST** con el URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

El contenido del mensaje se publica utilizando la serie de tema *topicString*.

Suscribir

Envíe una solicitud HTTP **DELETE** con el URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

El puente de WebSphere MQ para HTTP crea una suscripción no duradera gestionada para la serie de tema *topicString*. La suscripción se suprime tan pronto como se devuelve una publicación, o hasta que caduca el intervalo de espera establecido por la cabecera de entidad personalizada, `x-msg-wait`.

Ejecución del puente WebSphere MQ para ejemplos HTTP

El puente WebSphere MQ para ejemplos HTTP está disponible para su uso únicamente en el sistema operativo Windows . Los ejemplos muestran cómo enviar mandatos HTTP **POST** y HTTP **DELETE** al puente WebSphere MQ para HTTP desde programas Java.

Antes de empezar

Verifique el puente WebSphere MQ para la instalación HTTP ejecutando el paso "7" en la página 1046 en "Instalación, configuración y verificación del puente WebSphere MQ para HTTP" en la página 1045.

Los ejemplos HTTP se instalan en los directorios que se muestran en Tabla 156 en la página 1050. En cada caso, el código fuente se instala en el subdirectorio `/src` .

Tabla 156. Ubicación de ejemplos HTTP

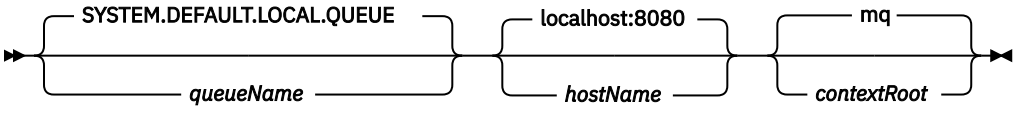
Plataforma	**Ubicación**
Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
Todas las demás plataformas	*MQ_INSTALLATION_PATH*/samp/http
MQ_INSTALLATION_PATH representa el directorio donde está instalado WebSphere MQ .	

Acerca de esta tarea

Los ejemplos simulan las aplicaciones de ejemplo WebSphere MQ AMQSPUT y AMQSGET. Ilustran las funciones siguientes en un entorno de mensajería punto a punto:

- **HTTPPOST** -Envía solicitudes HTTP **POST** en una aplicación Java para transferir mensajes a una cola WebSphere MQ , utilizando el puente WebSphere MQ para HTTP y maneja las respuestas.
- **HTTPDELETE** -Envía solicitudes HTTP **DELETE** en una aplicación Java para obtener mensajes de una cola WebSphere MQ , utilizando el puente WebSphere MQ para HTTP y maneja las respuestas que contienen el mensaje WebSphere MQ .

Parámetros para HTTPPOST y HTTPDELETE



Para ejecutar el ejemplo **HTTPPOST** , realice los pasos siguientes:

Procedimiento

1. En una ventana de mandatos, vaya al directorio de ejemplos HTTP.
2. Ejecute el ejemplo **HTTPPOST** .

```
java -classpath . HTTPPOST [parameters]
```

Cuando se inicia el ejemplo **HTTPPOST** , se muestra la salida siguiente:

```
HTTP POST Sample start
Target server is 'hostName'
Target queue is 'queueName'
Target context-root is 'contextRoot'
```

3. En el indicador de mandatos, escriba el texto que desea que forme el cuerpo del mensaje.
4. Pulse Intro para publicar el mensaje en la cola de WebSphere MQ .
   a) Si desea enviar otro mensaje, introduzca más texto.
      Los textos forman el cuerpo de un segundo mensaje WebSphere MQ .
   b) Pulse Intro para publicar el mensaje en la cola de WebSphere MQ .
5. Pulse Intro dos veces para finalizar **HTTPPOST**.
   Se visualiza la siguiente salida:

```
HTTP POST Sample end
```

Qué hacer a continuación

El ejemplo **HTTPDELETE** realiza una obtención destructiva de todos los mensajes que ha colocado en la cola de WebSphere MQ .

Ejecute el ejemplo **HTTPDELETE** completando los pasos siguientes:

1. En una ventana de mandatos, vaya a *MQ_INSTALLATION_PATH*/tools/samples. *MQ_INSTALLATION_PATH* representa el directorio donde está instalado WebSphere MQ .
2. Ejecute el ejemplo **HTTPDELETE** .

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   Cuando se inicia el ejemplo **HTTPDELETE** , se muestra la salida siguiente:

```
HTTP DELETE Sample start
Target server is 'host:port'
Target queue is 'your queue name'
Target context-root is 'your context-root'
message
message
...
```

Consideraciones de seguridad para el puente de WebSphere para HTTP

Las consideraciones de seguridad web estándar se aplican a la autenticación de un cliente de navegador web. La autorización para los recursos de WebSphere MQ está en el nivel del usuario que ejecuta el servlet WebSphere Bridge for HTTP y no el cliente de navegador web individual. La consideración de seguridad estándar de WebSphere MQ se aplica a WebSphere MQ.

Los datos que fluyen desde un navegador web a una aplicación WebSphere MQ utilizando el puente WebSphere para HTTP, y atrás, realiza tres pasos:

Conexión de cliente
Desde el navegador a WebSphere Bridge for HTTP sobre una conexión TCP/IP utilizando HTTP.

Conexión de adaptador de recursos a WebSphere MQ
La conexión es desde WebSphere Bridge for HTTP a un gestor de colas WebSphere MQ . La conexión es una conexión de cliente, a través de TCP/IP, o una conexión de enlaces WebSphere MQ local. Una vez realizada la conexión, la solicitud HTTP se coloca en una cola local estándar o en una cola de transmisión.

Desde la cola local de WebSphere MQ a través de uno o varios canales, hasta la cola de destino.
Aplique técnicas estándar para proteger colas, temas, gestores de colas y canales.

La respuesta realiza los pasos a la inversa.

Conexión de cliente

Conexiones seguras entre clientes HTTP y el servidor de aplicaciones utilizando el contenedor web. Utilice técnicas de servidor HTTP estándar, como por ejemplo el uso de HTTPS. Consulte la documentación del servidor de aplicaciones para obtener información.

Conexión de adaptador de recursos a WebSphere MQ

La conexión entre el adaptador de recursos y el gestor de colas está autorizada utilizando un único ID de usuario. Asigne un único ID de usuario para identificar solicitudes desde WebSphere Bridge for HTTP. El ID de usuario debe tener autorizaciones restringidas de WebSphere MQ sólo a los recursos a los que los usuarios externos deben tener acceso. Debe autenticar el cliente real por separado y establecer la confianza para las interacciones sucesivas con el cliente, utilizando técnicas estándar para la seguridad web.

Proteja la conexión entre el adaptador de recursos y el gestor de colas utilizando el ID de usuario único. Restrinja las autorizaciones que el ID de usuario no tiene a más de las necesarias para leer y escribir mensajes en colas y temas. WebSphere Bridge for HTTP es un punto de ataque entre Internet y la intranet.

La forma de proteger la conexión entre el adaptador de recursos y WebSphere MQ depende del adaptador de recursos específico. Consulte la documentación de para el adaptador de recursos.

Utilización de la interfaz de modelo de objeto de componente (WebSphere MQ Automation Classes for ActiveX)

Las clases de automatización de WebSphere MQ para ActiveX (MQAX) son componentes ActiveX que proporcionan clases que puede utilizar en la aplicación para acceder a WebSphere MQ.

MQAX requiere un entorno WebSphere MQ y una aplicación WebSphere MQ correspondiente con la que comunicarse.

Proporciona a la aplicación ActiveX la capacidad de ejecutar transacciones y acceder a datos en cualquiera de los sistemas empresariales a los que puede acceder a través de WebSphere MQ.

WebSphere MQ Automation Classes for ActiveX:

- Otorgue acceso a las funciones y características de la API WebSphere MQ , permitiendo una interconectividad completa con otras plataformas WebSphere MQ .
- Se ajustan a los convenios normales en los componentes de ActiveX.
- Se ajusta al modelo de objeto WebSphere MQ , también disponible para .NET, C++, Java y LotusScript.

Se proporcionan ejemplos para iniciar MQAX. Puede utilizar estos ejemplos inicialmente para comprobar que la instalación de MQAX se ha realizado correctamente y que tiene el entorno básico de WebSphere MQ en su lugar. Los ejemplos también muestran la forma de utilizar MQAX.

Creación y ejecución de scripts COM y ActiveX

El modelo de objeto de componente (COM) es un modelo de programación basado en objetos definido por Microsoft. Especifica cómo pueden suministrarse los componentes de software de forma que les permita localizar y comunicarse con todos los otros sin tener en cuenta del lenguaje informático en que se hayan escrito y de su ubicación.

ActiveX es un conjunto de tecnologías, basadas en COM, que integra el desarrollo de aplicaciones, componentes reutilizables y tecnologías de Internet en las plataformas Microsoft Windows . Los componentes de ActiveX proporcionan interfaces a las que las aplicaciones pueden acceder de forma dinámica. Un cliente de script de ActiveX es una aplicación, por ejemplo, un compilador, que puede crear o ejecutar un programa o script que utilice las interfaces proporcionadas por los componentes de ActiveX (o COM).

Soporte de entorno deWebSphere MQ

WebSphere MQ Automation Classes for ActiveX solo se puede utilizar con clientes de scripts **de 32 bits** ActiveX .

El componente COM sólo puede utilizarse para aplicaciones de **32 bits**. Si desea escribir una aplicación COM de 64 bits, puede utilizar la interfaz .NET.

Para ejecutar MQAX en un entorno de servidor WebSphere MQ debe tener instalado Windows 2000 o posterior en el sistema.

Para ejecutar MQAX en un entorno de cliente MQI de WebSphere MQ necesita un cliente MQI de WebSphere MQ en Windows 2000 o posterior instalado en el sistema:

El cliente MQI de WebSphere MQ requiere acceso a al menos un servidor WebSphere MQ . Cuando el cliente MQI de WebSphere MQ y el servidor de WebSphere MQ están instalados en el sistema, las aplicaciones MQAX siempre se ejecutan en el servidor. La interfaz ActiveX para la MQAI solo está disponible en entornos de servidor WebSphere MQ .

Diseño y programación utilizando WebSphere MQ Automation Classes for ActiveX

Diseño de aplicaciones MQAX que acceden a aplicaciones que no son de ActiveX

Las clases de automatización de WebSphere MQ proporcionan acceso a las funciones de la API de WebSphere MQ . Por lo tanto, puede beneficiarse de todas las ventajas que el uso de WebSphere MQ puede aportar a la aplicación Windows .

El diseño general de la aplicación es el mismo que para cualquier aplicación de WebSphere MQ , por lo que debe tener en cuenta todos los aspectos de diseño descritos en la sección “Desarrollo de aplicaciones” en la página 7 .

Para utilizar las clases de automatización de WebSphere MQ , codifique los programas Windows en la aplicación utilizando un lenguaje que dé soporte a la creación y el uso de objetos COM. Por ejemplo, Visual Basic, Java y otros clientes de scripts ActiveX . A continuación, las clases se pueden integrar fácilmente en la aplicación porque los objetos de WebSphere MQ que necesita se pueden codificar utilizando la sintaxis nativa del lenguaje de implementación.

Utilización de WebSphere MQ Automation Classes for ActiveX

Al diseñar una aplicación ActiveX que utiliza WebSphere MQ Automation Classes for ActiveX, el elemento de información más importante es el mensaje que se envía o recibe desde el sistema WebSphere MQ remoto. Por lo tanto, hay que conocer el formato de los elementos que se insertan en el mensaje. Para un script MQAX a un trabajo, tanto él como la aplicación WebSphere MQ que recoge o envía el mensaje deben conocer la estructura del mensaje.

Si va a enviar un mensaje con una aplicación MQAX y desea realizar una conversión de datos en el extremo de MQAX, también ha de saber:

- La página de códigos utilizada por el sistema remoto.
- La codificación utilizada por el sistema remoto.

Para mantener su código portable, es una buena práctica establecer la página de códigos y la codificación, incluso si en ese momento son los mismos en los sistemas de envío y recepción.

Al considerar cómo estructurar la implementación del sistema que diseña, recuerde que los scripts MQAX se ejecutan en la misma máquina en la que tiene instalado el gestor de colas WebSphere MQ o el cliente WebSphere MQ .

Consejos y sugerencias de programación

El orden en el que se presentan estos consejos y sugerencias no es significativo. Son temas que, si son adecuados para el trabajo que está efectuando, pueden ahorrarle tiempo.

Propiedades del descriptor de mensaje

Si manipula propiedades de descriptor de mensaje en un programa, puede que sea mejor utilizar los equivalentes en hexadecimal de los campos.

La información de esta sección hace referencia a las propiedades siguientes:

- AccountingToken
- CorrelationId
- GroupId
- MessageId

Cuando una aplicación WebSphere MQ es el originador de un mensaje y WebSphere MQ genera estas propiedades, es mejor utilizar las propiedades Hex de AccountingToken, CorrelationIdHex, GroupIdHex y MessageIdHex si desea ver sus valores o manipularlos de cualquier forma, incluyendo su devolución en un mensaje a WebSphere MQ. La razón de esto es que los valores generados de WebSphere MQ son series de bytes que tienen cualquier valor de 0 a 255 inclusive, no son series de caracteres imprimibles.

Cuando el script MQAX sea el que origina un mensaje, puede utilizar las propiedades AccountingToken, CorrelationId, GroupId y MessageId o sus equivalentes en hexadecimal.

Constantes de WebSphere MQ

Las constantes de WebSphere MQ se proporcionan como miembros de la enumeración WebSphere MQ en la biblioteca MQAX200.

Constantes de serie de WebSphere MQ

WebSphere MQ constantes de serie y sus series de caracteres correspondientes.

Las constantes de serie de WebSphere MQ no están disponibles cuando se utiliza WebSphere MQ Automation Classes for ActiveX. Debe utilizar la serie de caracteres explícita para aquellos que se muestran en la lista siguiente y para cualquier otro que necesite. Los mandatos deben rellenarse usando espacios hasta que tengan ocho caracteres:

MQFMT_NONE	" "
MQFMT_ADMIN	"MQADMIN "
MQFMT_CHANNEL_COMPLETED	"MQCHCOM "
MQFMT_CICS	"MQCICS "
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD "
MQFMT_DIST_HEADER	"MQHDIST "
MQFMT_EVENT	"MQEVENT "
MQFMT_IMS	"MQIMS "
MQFMT_IMS_VAR_STRIsNG	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "

MQFMT_REF_MSG_HEADER	"MQHREF "
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

Constantes de series de caracteres nulas

Las constantes de WebSphere MQ , utilizadas para la inicialización de cuatro propiedades MQMessage, MQMI_NONE (24 caracteres NULL), MQCI_NONE (24 caracteres NULL), MQGI_NONE (24 caracteres NULL) y MQACT_NONE (32 caracteres NULL), no están soportadas por WebSphere MQ Automation Classes for ActiveX. Darles el valor de series de caracteres vacías surte el mismo efecto.

Por ejemplo, para establecer los diversos ID de un MQMessage en estos valores: *mymessage*.**MessageId** = "" *mymessage*.**CorrelationId** = "" *mymessage*.**AccountingToken** = ""

Recepción de un mensaje de WebSphere MQ

Hay varias formas de recibir un mensaje de WebSphere MQ:

- Efectuando un sondeo emitiendo una llamada GET seguida de Wait y utilizando la función TIMER de Visual Basic.
- Emitiendo una llamada GET con la opción Wait; la duración de la espera se especifica definiendo la propiedad WaitInterval. Tenga en cuenta esto cuando, aunque configure el sistema para su ejecución en un entorno de varias hebras, el software que se ejecuta en este momento solo se pueda ejecutar en una sola hebra. Esto evita que el sistema se bloquee indefinidamente.

  Esto no afectará a la operación de otras hebras. Sin embargo, si las otras hebras requieren acceso a WebSphere MQ, requieren una segunda conexión con WebSphere MQ utilizando objetos de cola y gestor de colas MQAX adicionales.

La emisión de una operación GET con la opción Wait y el establecimiento de WaitInterval en MQWI_UNLIMITED provoca que el sistema se bloquee hasta que finalice la llamada GET, si el proceso es de una sola hebra.

Utilización de la conversión de datos

WebSphere MQ Automation Classes for ActiveX da soporte a dos formas de conversión de datos: codificación numérica y conversión de juegos de caracteres.

Codificación numérica

Si define la propiedad Encoding de MQMessage, los siguientes métodos efectuarán conversiones entre los distintos sistemas de codificación numérica:

- Método ReadDecimal2
- Método ReadDecimal4
- Método ReadDouble
- Método ReadDouble4
- Método ReadFloat
- Método ReadInt2
- Método ReadInt4
- Método ReadLong

- Método ReadShort
- Método ReadUInt2
- Método WriteDecimal2
- Método WriteDecimal4
- Método WriteDouble
- Método WriteDouble4
- Método WriteFloat
- Método WriteInt2
- Método WriteInt4
- Método WriteLong
- Método WriteShort
- Método WriteUInt2

La propiedad Codificación se puede establecer e interpretar utilizando las constantes de WebSphere MQ proporcionadas. Figura 216 en la página 1056 muestra un ejemplo de estos:

```
/* Encodings for Binary Integers */
        MQENC_INTEGER_UNDEFINED
        MQENC_INTEGER_NORMAL
        MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
        MQENC_DECIMAL_UNDEFINED
        MQENC_DECIMAL_NORMAL
        MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
        MQENC_FLOAT_UNDEFINED
        MQENC_FLOAT_IEEE_NORMAL
        MQENC_FLOAT_IEEE_REVERSED
        MQENC_FLOAT_S390
```

Figura 216. Constantes de WebSphere MQ proporcionadas para la codificación

Por ejemplo, para enviar un entero desde un sistema Intel a un sistema operativo System/390 en la codificación System/390 :

```
Dim msg As New MQMessage   'Define a WebSphere MQ message for our use..
Print msg.Encoding    'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg.Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg.Encoding    'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg.WriteLong(local_num)         'Write the number into the message
```

Conversión de juegos de caracteres

La conversión de juegos de caracteres es necesaria cuando se envía un mensaje de un sistema a otro en el que las páginas de códigos son diferentes. La conversión de páginas de códigos la utilizan:

- método ReadString
- Método ReadNullTerminatedString
- Método WriteString
- Método WriteNullTerminatedString
- Propiedad MessageData

Hay que establecer la propiedad CharacterSet de MQMessage a un juego de caracteres soportado (CCSID).

WebSphere MQ Automation Classes for ActiveX utiliza tablas de conversión para realizar la conversión del juego de caracteres.

Por ejemplo, para convertir cadenas automáticamente a la página de códigos 437:

```
Dim msg As New MQMessage                 'Define a WebSphere MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

El método WriteString recibe los datos de serie ("Una serie de caracteres" en el ejemplo) como una serie Unicode. Después convierte los datos Unicode en datos de la página de códigos 437 utilizando la tabla de conversión 34B001B5.TBL.

Los caracteres de la cadena de caracteres Unicode que no están soportados en la página de códigos 437 reciben el carácter de sustitución estándar de la página de códigos 437.

De forma similar, cuando se utiliza el método ReadString , el mensaje de entrada tiene un juego de caracteres establecido por el valor de WebSphere MQ Message Descriptor (MQMD) y hay una conversión de esta página de códigos a Unicode antes de que se vuelva a pasar al lenguaje de scripts.

Generación de hebras

WebSphere MQ Automation Classes for ActiveX implementa un modelo de hebras libres donde se pueden utilizar objetos entre hebras.

Aunque MQAX permite el uso de objetos MQQueue y MQQueueManager , WebSphere MQ no permite actualmente el uso compartido de descriptores de contexto entre distintas hebras.

Los intentos de utilizarlos en otra hebra dan como resultado un error y WebSphere MQ devuelve un código de retorno de MQRC_HCONN_ERROR.

Nota: Solo hay un objeto MQSession por proceso. No es aconsejable usar los CompletionCode y ReasonCode de MQSession en entornos de varias hebras. Es posible que una segunda hebra sobrescriba los valores de error de MQSession en el intervalo que transcurre entre la elevación de un error y su comprobación en la primera hebra. Las hebras se serializan mientras dura cada llamada de método o acceso de propiedad. Por lo tanto, la emisión de una opción de obtención con espera hace que otras hebras que acceden a objetos MQAX se suspendan hasta que se complete la operación.

Tratamiento de errores

Esta información describe las propiedades de un objeto MQAX, cómo funciona el tratamiento de errores, las reglas que describen cómo se maneja la generación de excepciones y la obtención de una propiedad.

Cada objeto MQAX incluye propiedades para guardar información de error y un método para restablecerlas o borrarlas. Las propiedades son:

- CompletionCode
- ReasonCode
- ReasonName

El método es:

- ClearErrorCodes

Cómo funciona el tratamiento de errores

El script de MQAX o la aplicación invocan el método de un objeto MQAX, o acceden o actualizan una propiedad de dicho objeto MQAX:

1. Se actualizan las propiedades ReasonCode y CompletionCode del objeto en cuestión.

2. Las propiedades ReasonCode y CompletionCode del objeto MQSession también se actualizan con la misma información.

   Nota: Para obtener información sobre las limitaciones de uso de los códigos de error de MQSession en aplicaciones con hebras, consulte “Generación de hebras” en la página 1057.

   Si CompletionCode es mayor que o igual que la propiedad ExceptionThreshold de MQSession, MQAX genera una excepción (número 32000). Procésela en el script con la sentencia On Error (o equivalente).
3. Use la función Error para recuperar la cadena de error asociada, que tiene este formato:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   Para obtener más información sobre cómo utilizar las sentencias On Error, consulte la documentación del lenguaje de script ActiveX.

   La utilización de CompletionCode y ReasonCode del objeto MQSession resulta cómoda en los manejadores de errores simples.

   La propiedad ReasonName devuelve el nombre simbólico WebSphere MQ para el valor actual de ReasonCode.

Generación de excepciones

Las reglas siguientes describen cómo se manejan las generaciones de excepciones:

- Siempre que una propiedad o un método establece el código de terminación a un valor mayor o igual que el umbral de excepción (que suele establecerse a 2), se genera una excepción.
- Todas las llamadas de método y los conjuntos de propiedades establecen el código de terminación.

Obtención de una propiedad

Este es un caso especial, porque CompletionCode y ReasonCode no siempre se actualizan:

- Si la obtención de una propiedad es correcta, el ReasonCode y el CompletionCode del objeto y de MQSession no varían.
- Si la obtención de una propiedad falla con un CompletionCode de advertencia, el ReasonCode y el CompletionCode no varían.
- Si la obtención de una propiedad falla con un CompletionCode de error, el ReasonCode y el CompletionCode se actualizan para reflejar los verdaderos valores y se procede con el procesamiento del error tal y como se ha descrito.

La clase MQSession tiene un método *ReasonCodeName* que se puede utilizar para sustituir un código de razón WebSphere MQ por un nombre simbólico. Esto es especialmente útil cuando se desarrollan programas en los que se pueden producir errores inesperados. Sin embargo, el nombre no es apto para presentárselo a los usuarios.

Cada clase también tiene una propiedad *ReasonName*, que devuelve el nombre simbólico del código de razón actual de dicha clase.

Referencia de WebSphere MQ Automation Classes for ActiveX

En esta sección se describen las clases de WebSphere MQ Automation Classes for ActiveX (MQAX), desarrolladas para ActiveX. Las clases le permiten escribir aplicaciones ActiveX que pueden acceder a otras aplicaciones que se ejecutan en entornos que no sonActiveX , utilizando WebSphere MQ.

Interfaz de WebSphere MQ Automation Classes for ActiveX

WebSphere MQ Automation Classes for ActiveX proporciona constantes ActiveX numéricas predefinidas (como MQMT_REQUEST) necesarias para utilizar las clases.

Las clases de automatización de ActiveX constan de lo siguiente:

- “clase MQSession” en la página 1060
- “clase MQQueueManager” en la página 1063
- “clase MQQueue” en la página 1074
- “Clase MQMessage” en la página 1089
- “Clase MQPutMessageOptions” en la página 1110
- “clase MQGetMessageOptions” en la página 1113
- “Clase MQDistributionList” en la página 1115
- “Clase MQDistributionListItem” en la página 1119

Además, WebSphere MQ Automation Classes for ActiveX proporciona constantes numéricas ActiveX predefinidas (como MQMT_REQUEST) necesarias para utilizar las clases. Estas se proporcionan en la enum de MQ de la biblioteca MQAX200. Las constantes son un subconjunto de las definidas en los archivos de cabecera C de WebSphere MQ (cmqc * .h) con algunos códigos de razón adicionales de WebSphere MQ Automation Classes for ActiveX .

Acerca de las clases de WebSphere MQ Automation Classes for ActiveX

Lea esta información junto con los temas de referencia en Desarrollo de referencias de aplicaciones.

Consulte Características que solo se pueden utilizar con la instalación primaria en Windows para obtener información importante.

La clase MQSession proporciona un objeto raíz que contiene el estado de la última acción realizada en cualquiera de los objetos de MQAX. Consulte “Tratamiento de errores” en la página 1057 para obtener más información.

Las clases MQQueueManager y MQQueue proporcionan acceso a los objetos WebSphere MQ subyacentes. Los métodos o accesos de propiedad para estas clases en general dan como resultado que se realicen llamadas a través de la MQI de WebSphere MQ .

Las clases MQMessage, MQPutMessageOptions y MQGetMessageOptions encapsulan las estructuras de datos MQMD, MQPMO y MQGMO, y son una ayuda para enviar mensajes a colas y recuperar mensajes de ellas.

La clase MQDistributionList encapsula un grupo de colas (local, remota o de alias) para salida. La clase MQDistributionListItem encapsula las estructuras MQOR, MQRR y MQPMR, y las asocia a una lista de distribución propietaria.

Paso de parámetros

En las invocaciones de métodos, todos los parámetros se pasan por valor, excepto cuando el parámetro es un objeto, en cuyo caso lo que se pasa es la referencia.

Las definiciones de clase proporcionadas listan el tipo de datos de cada parámetro o propiedad. En muchos clientes de ActiveX como, por ejemplo, Visual Basic, si la variable utilizada no es del tipo requerido, el valor se convierte automáticamente a, o desde, el tipo necesario, siempre que dicha conversión sea posible. Esto sigue las reglas estándar del cliente; MQAX no proporciona dicha conversión.

Muchos de los métodos reciben parámetros de cadena de longitud fija o devuelven una cadena de caracteres de longitud fija. Las reglas de conversión son las siguientes:

- Si el usuario proporciona una cadena cuya longitud fija es incorrecta parámetro de entrada o valor de retorno, el valor se trunca o se rellena con los espacios finales según proceda.
- Si el usuario proporciona como parámetro de entrada una cadena de longitud variable cuya longitud es incorrecta, el valor se trunca o se rellena con espacios finales.

- Si el usuario facilita como valor de retorno una cadena de caracteres de longitud variable cuya longitud es errónea, dicha cadena se ajusta para adoptar la longitud requerida (porque devolver un valor destruye de todos modos el valor anterior de la cadena de caracteres).
- Las cadenas proporcionadas como parámetros de entrada pueden incorporar nulos.

Estas clases se encuentran en la biblioteca MQAX200.

Métodos de acceso a objetos

Estos métodos no están directamente relacionados con ninguna llamada WebSphere MQ . Cada uno de estos métodos crea un objeto en el que se retiene la información de referencia, seguido de conectarse o abrir un objeto WebSphere MQ :

Cuando se realiza una conexión con un gestor de colas, contiene el atributo 'descriptor de conexión' generado por WebSphere MQ.

Cuando se abre una cola, contiene el atributo 'descriptor de objeto' generado por WebSphere MQ.

Estos atributos de WebSphere MQ no están directamente disponibles para el programa MQAX.

Errores

El cliente ActiveX puede detectar errores sintácticos en el paso de parámetros en tiempo de compilación y de ejecución. Los errores pueden capturarse con On Error en Visual Basic.

Las clases de WebSphere MQ ActiveX contienen todas dos propiedades especiales de sólo lectura: ReasonCode y CompletionCode. Estas propiedades se pueden leer en cualquier momento.

Un intento de acceder a cualquier otra propiedad o de emitir cualquier llamada de método puede generar un error desde WebSphere MQ.

Si un conjunto de propiedades o una invocación de método son satisfactorias, el objeto ReasonCode del objeto propietario se establece a MQRC_NONE y CompletionCode se establece a MQCC_OK.

Si el acceso a propiedad o la invocación de método no son satisfactorios, en los campos se definen códigos de razón y de terminación.

clase MQSession

Esta es la clase raíz para WebSphere MQ Automation Classes for ActiveX.

Hay siempre un solo objeto MQSession por proceso de cliente ActiveX. Si se intenta crear un segundo objeto, se creará una segunda referencia al objeto original.

Creación

New crea un objeto MQSession nuevo.

Sintaxis

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

Propiedades

- “Propiedad CompletionCode” en la página 1061.
- “Propiedad ExceptionThreshold” en la página 1061.
- “propiedad ReasonCode” en la página 1061.
- “propiedad ReasonName” en la página 1062.

Método

- “Método AccessGetMessageOptions” en la página 1062.
- “Método AccessMessage” en la página 1062.
- “Método AccessPutMessageOptions” en la página 1062.
- “Método AccessQueueManager” en la página 1062.
- “Método ClearErrorCodes” en la página 1063.
- “Método ReasonCodeName” en la página 1063.

Propiedad CompletionCode

Solo lectura. Devuelve el código de finalización WebSphere MQ establecido por el método o conjunto de propiedades más reciente emitido para cualquier objeto WebSphere MQ .

Vuelve a adoptar el valor MQCC_OK cuando se invoca satisfactoriamente un conjunto de métodos o propiedades contra cualquier objeto MQAX.

Un manejador de sucesos de error puede revisar esta propiedad para diagnosticar el error sin necesidad de saber qué objeto estaba implicado en la operación.

La utilización de CompletionCode y ReasonCode en el objeto MQSession es muy conveniente para manejadores de error sencillos.

Nota: Para obtener información sobre las limitaciones de uso de los códigos de error de MQSession en aplicaciones con hebras, consulte “Generación de hebras” en la página 1057.

Definido en: clase MQSession

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis:

Para obtener: *completioncode & = MQSession*.CompletionCode

Propiedad ExceptionThreshold

Lectura-grabación. Define el nivel de error de WebSphere MQ para el que MQAX emitirá una excepción. Toma como valor predeterminado MQCC_FAILED. Un valor mayor que MQCC_FAILED evita de forma efectiva el procesamiento de excepciones, delegando en el programador las comprobaciones de CompletionCode y ReasonCode.

Definido en: clase MQSession

Tipo de datos: Long

Valores:

- Cualquiera, pero considere MQCC_WARNING, MQCC_FAILED o superiores.

Sintaxis:

Para obtener: *ExceptionThreshold& = MQSession*.**ExceptionThreshold**

Para establecer: *MQSession.***ExceptionThreshold** = *ExceptionThreshold$*

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el método o conjunto de propiedades más reciente emitido para cualquier objeto WebSphere MQ .

Un manejador de sucesos de error puede revisar esta propiedad para diagnosticar el error sin necesidad de saber qué objeto estaba implicado en la operación.

La utilización de CompletionCode y ReasonCode en el objeto MQSession es muy conveniente para manejadores de error sencillos.

Nota: Para obtener información sobre las limitaciones de uso de los códigos de error de MQSession en aplicaciones con hebras, consulte "Generación de hebras" en la página 1057.

Definido en: clase MQSession

Tipo de datos: Long

Valores:

- Consulte Razón (MQLONG) y los valores de MQAX adicionales que se listan en "códigos de razón" en la página 1126.

Sintaxis: Para obtener: *reasoncode & = MQSession*.**ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Nota: Para obtener información sobre las limitaciones de uso de los códigos de error de MQSession en aplicaciones con hebras, consulte "Generación de hebras" en la página 1057.

Definido en: clase MQSession

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQSession*.**ReasonName**

Método AccessGetMessageOptions

Crea un objeto MQGetMessageOptions.

Definido en: clase MQSession

Sintaxis: *gmo = MQSession*.**AccessGetMessageOptions()**

Método AccessMessage

Crea un objeto MQMessage.

Definido en: clase MQSession

Sintaxis: *msg = MQSession*.**AccessMessage()**

Método AccessPutMessageOptions

Crea un objeto MQPutMessageOptions.

Definido en: clase MQSession

Sintaxis: *pmo = MQSession*.**AccessPutMessageOptions()**

Método AccessQueueManager

Crea un nuevo objeto MQQueueManager y lo conecta a un gestor de colas real mediante el cliente MQI de WebSphere MQ o el servidor WebSphere MQ . Además de realizar una conexión, este método también realiza una apertura para el objeto gestor de colas.

Cuando el cliente MQI de WebSphere MQ y el servidor WebSphere MQ están instalados en el sistema, las aplicaciones MQAX se ejecutarán en el servidor de forma predeterminada. Para ejecutar MQAX en el

cliente, la biblioteca de enlaces de cliente debe especificarse en la variable de entorno `GMQ_MQ_LIB` , por ejemplo, establezca `GMQ_MQ_LIB=mqic.dll`.

En una instalación de solo cliente, no es necesario establecer la variable de entorno `GMQ_MQ_LIB`. Cuando esta variable no está establecida, WebSphere MQ intenta cargar amqzst.dll. Si esta DLL no está presente (como es el caso en una instalación sólo de cliente), WebSphere MQ intenta cargar mqic.dll.

Si la operación es satisfactoria, establece el ConnectionStatus de MQQueueManager a TRUE.

Un gestor de colas puede conectarse, como máximo, con un único objeto MQQueueManager por instancia ActiveX.

Si la conexión con el gestor de colas falla, se genera un suceso de error y se establecen los ReasonCode y CompletionCode del objeto MQSession.

Definido en: clase MQSession

Sintaxis: *set qm = MQSession*.**AccessQueueManager** (*Nombre$*)

Parámetro:*Name$* String. Nombre del gestor de colas con el que se va a conectar.

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE.

Definido en: clase MQSession

Sintaxis: Llamar a *MQSession*.**ClearErrorCodes ()**

Método ReasonCodeName

Devuelve el nombre del código de razón con el valor numérico dado. Resulta útil para proporcionar indicaciones más claras de condiciones de error a los usuarios. El nombre sigue siendo algo críptico (por ejemplo, ReasonCodeName (2059) es **MQRC_Q_MGR_NOT_AVAILABLE**), por lo que los posibles errores deben capturarse y sustituirse por texto descriptivo adecuado para la aplicación.

Definido en: clase MQSession

Sintaxis: *errname $= MQSession*.**ReasonCodeName**(*reasonCode&*)

Parámetro:*reasoncode &* Long. El código de razón cuyo nombre simbólico se requiere.

clase MQQueueManager

Esta clase representa una conexión con un gestor de colas. El gestor de colas puede estar ejecutándose localmente (un servidor WebSphere MQ) o de forma remota con acceso proporcionado por el cliente WebSphere MQ . Una aplicación debe crear un objeto de esta clase y conectarlo a un gestor de colas. Cuando un objeto de esta clase se destruye, se desconecta automáticamente de su gestor de colas.

Contención

Los objetos de clase MQQueue están asociados a esta clase.

New crea un nuevo objeto MQQueueManager y devuelve todas las propiedades a sus valores iniciales. De forma alternativa, también pueden utilizar el método AccessQueueManager de la clase MQSession.

Creación

New crea un **nuevo** objeto MQQueueManager y devuelve todas las propiedades a sus valores iniciales. De forma alternativa, también pueden utilizar el método AccessQueueManager de la clase MQSession.

Sintaxis

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

Propiedades

- “Propiedad AlternateUserId” en la página 1065.
- “Propiedad AuthorityEvent” en la página 1065.
- “Propiedad BeginOptions” en la página 1065.
- “propiedad ChannelAutoDefinition” en la página 1066.
- “Propiedad ChannelAutoDefinitionEvent” en la página 1066.
- “propiedad ChannelAutoDefinitionExit” en la página 1066.
- “propiedad CharacterSet” en la página 1066.
- “Propiedad CloseOptions” en la página 1067.
- “Propiedad CommandInputQueueName” en la página 1067.
- “Propiedad CommandLevel” en la página 1067.
- “Propiedad CompletionCode” en la página 1067.
- “Propiedad ConnectionHandle” en la página 1067.
- “Propiedad ConnectionStatus” en la página 1068.
- “Propiedad ConnectOptions” en la página 1068.
- “Propiedad DeadLetterQueueName” en la página 1068.
- “Propiedad DefaultTransmissionQueueName” en la página 1068.
- “Propiedad Description” en la página 1068.
- “propiedad DistributionLists” en la página 1068.
- “Propiedad InhibitEvent” en la página 1069.
- “Propiedad IsConnected” en la página 1069.
- “propiedad IsOpen” en la página 1069.
- “Propiedad LocalEvent” en la página 1069.
- “Propiedad MaximumHandles” en la página 1070.
- “Propiedad MaximumMessageLength” en la página 1070.
- “Propiedad MaximumPriority” en la página 1070.
- “Propiedad MaximumUncommittedMessages” en la página 1070.
- “Propiedad Name” en la página 1070.
- “Propiedad ObjectHandle” en la página 1070.
- “Propiedad PerformanceEvent” en la página 1071.
- “Propiedad Platform” en la página 1071.
- “propiedad ReasonCode” en la página 1071.
- “propiedad ReasonName” en la página 1071.
- “Propiedad RemoteEvent” en la página 1071.
- “Propiedad StartStopEvent” en la página 1072.
- “Propiedad SyncPointAvailability” en la página 1072.
- “Propiedad TriggerInterval” en la página 1072.

Métodos

- “Método AccessQueue” en la página 1072.
- “Método AddDistributionList” en la página 1073.
- “Método Backout” en la página 1073.
- “Método Begin” en la página 1073.

- “Método ClearErrorCodes” en la página 1073.
- “Método Commit” en la página 1074.
- “Método Connect” en la página 1074.
- “Método Disconnect” en la página 1074.

Acceso de propiedad

Se puede acceder a las propiedades siguientes en cualquier momento.

- “Propiedad AlternateUserId” en la página 1065.
- “Propiedad CompletionCode” en la página 1067.
- “Propiedad ConnectionStatus” en la página 1068.
- “propiedad ReasonCode” en la página 1071.

Al resto de las propiedades solo se puede acceder si el objeto está conectado a un gestor de colas y el ID de usuario tiene autorización para efectuar consultas para dicho gestor de colas. Si se establece un ID de usuario alternativo y el ID de usuario actual tiene autorización para utilizarlo, en su lugar, se comprueba si el ID de usuario está autorizado a realizar consultas.

Si estas condiciones no se aplican, WebSphere MQ Automation Classes for ActiveX intenta conectarse al gestor de colas y abrirlo para realizar consultas automáticamente. Si esto no se realiza correctamente, la llamada establece un CompletionCode de MQCC_FAILED y uno de los siguientes ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

Propiedad AlternateUserId

Lectura-grabación. El ID de usuario alternativo que se utiliza para validar el acceso a los atributos del gestor de colas.

Esta propiedad no se puede establecer si IsConnected es TRUE.

Esta propiedad no se puede establecer mientras el objeto está abierto.

`Defined in:` clase MQQueueManager

`Data Type:` Serie de 12 caracteres

`Syntax:` Para obtener: *altuser $= MQQueueManager*.**AlternateUserId** Para establecer: *MQQueueManager*.**AlternateUserId** = *altuser $*

Propiedad AuthorityEvent

Solo lectura. El atributo AuthorityEvent de la MQI.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *authevent = MQQueueManager*.**AuthorityEvent**

Propiedad BeginOptions

Lectura-grabación. Son las opciones que se aplican al método Begin. Inicialmente MQBO_NONE.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQBO_NONE

Sintaxis: Para obtener: *beginoptions & =MQQueueManager*.**BeginOptions**

Para establecer: *MQQueueManager*.**BeginOptions**=*beginoptions &*

propiedad ChannelAutoDefinition

Solo lectura. Controla si está permitida la definición automática de canales.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQCHAD_DISABLED
- MQCHAD_ENABLED

Sintaxis: Para obtener: *channelautodef & = MQQueueManager*.**ChannelAutoDefinition**

Propiedad ChannelAutoDefinitionEvent

Solo lectura. Controla si se generan sucesos de definición automática de canales.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *channelautodefevent & =MQQueueManager*.**ChannelAutoDefinitionEvent**

propiedad ChannelAutoDefinitionExit

Solo lectura. El nombre de la salida de usuario que se utiliza para la definición de canal automática.

Definida en:
clase MQQueueManager

Tipo de datos:
Cadena

Sintaxis: Para obtener: *channelautodefexit$= MQQueueManager*.**ChannelAutoDefinitionExit**

propiedad CharacterSet

Solo lectura. Es el atributo CodedCharSetId de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *characterset & = MQQueueManager*.**CharacterSet**

Propiedad CloseOptions

Lectura-grabación. Opciones que se utilizan para controlar lo que sucede cuando se cierra el gestor de colas. El valor inicial es MQCO_NONE.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQCO_NONE

Sintaxis: Para obtener: *closeopt & = MQQueueManager*.**CloseOptions**

Para establecer: *MQQueueManager*.**CloseOptions** =*closeopt &*

Propiedad CommandInputQueueName

Solo lectura. Es el atributo CommandInputQName de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *commandinputqname $= MQQueueManager*.**CommandInputQueueName**

Propiedad CommandLevel

Solo lectura. Devuelve la versión y el nivel de la implementación del gestor de colas de WebSphere MQ (atributo CommandLevel de MQI)

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *level & = MQQueueManager*.**CommandLevel**

Propiedad CompletionCode

Solo lectura. Devuelve el código de terminación establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = MQQueueManager*.**CompletionCode**

Propiedad ConnectionHandle

Solo lectura. El descriptor de conexión para el objeto de gestor de colas WebSphere MQ .

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Sintaxis: Para obtener: *hconn & = MQQueueManager.***ConnectionHandle**

Propiedad ConnectionStatus

Solo lectura. Indica si el objeto se conecta o no a su gestor de colas.

Definido en: Clase MQQueueManager

Tipo de datos: Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *status = MQQueueManager***.ConnectionStatus**

Propiedad ConnectOptions

Lectura-Escritura. Estas opciones se aplican al método Connect. Inicialmente, MQCNO_NONE.

Definida en:
clase MQQueueManager

Tipo de datos:
Long

Valores:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

Sintaxis: Para obtener: *connectoptions & =MQQueueManager*.**ConnectOptions**

Para establecer: *MQQueueManager***.ConnectOptions**=*connectoptions &*

Propiedad DeadLetterQueueName

Solo lectura. Es el atributo DeadLetterQName de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: String de 48 caracteres

Sintaxis: para obtener: *dlqname $= MQQueueManager***.DeadLetterQueueName**

Propiedad DefaultTransmissionQueueName

Solo lectura. Es el atributo DefXmitQName de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *defxmitqname $= MQQueueManager***.DefaultTransmissionQueueName**

Propiedad Description

Solo lectura. Es el atributo QMgrDesc de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: String de 64 caracteres

Sintaxis: Para obtener: *description $= MQQueueManager***.Descripción**

propiedad DistributionLists

Solo lectura. Esta es la función del gestor de colas para dar soporte a listas de distribución.

Definida en:
clase MQQueueManager

Tipo de datos:
Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *distributionlists= MQQueueManager*.**DistributionLists**

Propiedad InhibitEvent

Solo lectura. Es el atributo InhibitEvent de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *inhibevent & = MQQueueManager*.**InhibitEvent**

Propiedad IsConnected

Valor que indica si el gestor de colas está conectado en ese momento.

Solo lectura.

Definido en: Clase MQQueueManager

Tipo de datos: Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *isconnected = MQQueueManager*.**IsConnected**

propiedad IsOpen

Un valor que indica si el gestor de colas está abierto para consultas actualmente.

Solo lectura.

Definida en:
clase MQQueueManager

Tipo de datos:
Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *IsOpen = MQQueueManager*.**IsOpen**

Propiedad LocalEvent

Solo lectura. Es el atributo LocalEvent de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *localevent & = MQQueueManager***.LocalEvent**

Propiedad MaximumHandles

Solo lectura. Es el atributo MaxHandles de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *maxhandle & = MQQueueManager***.MaximumHandles**

Propiedad MaximumMessageLength

Solo lectura. Es el atributo MaxMsgLength del gestor de colas de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *maxmessagelength & = MQQueueManager***.MaximumMessageLength**

Propiedad MaximumPriority

Solo lectura. Es el atributo MaxPriority de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *maxpriority & = MQQueueManager***.MaximumPriority**

Propiedad MaximumUncommittedMessages

Solo lectura. Es el atributo MaxUncommittedMsgs de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *maxuncommitted & = MQQueueManager***.MaximumUncommittedMensajes**

Propiedad Name

Lectura-grabación. Atributo QMgrName de la MQI. Esta propiedad no se puede escribir una vez conectado el MQQueueManager.

Definido en: Clase MQQueueManager

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *name $= MQQueueManager***.name**

Para establecer: *MQQueueManager***.name** = *nombre $*

Nota: Visual Basic reserva la propiedad "Name" para su uso en la interfaz visual. Por lo tanto, cuando lo use en Visual Basic, emplee minúsculas, es decir, "name".

Propiedad ObjectHandle

Solo lectura. El descriptor de contexto de objeto para el objeto de gestor de colas WebSphere MQ .

Definida en:
clase MQQueueManager

Tipo de datos
Long

Sintaxis: Para obtener: *hobj & = MQQueueManager.***ObjectHandle**

Propiedad PerformanceEvent

Solo lectura. Es el atributo PerformanceEvent de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *perfevent & = MQQueueManager*.PerformanceEvent

Propiedad Platform

Solo lectura. Es el atributo Platform de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

Sintaxis: Para obtener: *platform & = MQQueueManager***.Plataforma**

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = MQQueueManager***.ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido en: Clase MQQueueManager

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQQueueManager***.ReasonName**

Propiedad RemoteEvent

Solo lectura. Es el atributo RemoteEvent de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *remoteevent & = MQQueueManager*.**RemoteEvent**

Propiedad StartStopEvent

Solo lectura. Es el atributo StartStopEvent de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *strstpevent & = MQQueueManager*.**StartStopSuceso**

Propiedad SyncPointAvailability

Solo lectura. Es el atributo SyncPoint de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Valores:

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

Sintaxis: Para obtener: *syncpuntutavailability & = MQQueueManager*.**SyncPointAvailability**

Propiedad TriggerInterval

Solo lectura. Es el atributo TriggerInterval de la MQI.

Definido en: Clase MQQueueManager

Tipo de datos: Long

Sintaxis: Para obtener: *trigint & = MQQueueManager*.**TriggerInterval**

Método AccessQueue

Crea un objeto MQQueue y lo asocia a este objeto MQQueueManager estableciendo la propiedad de referencia de conexión de la cola. Asigna a las propiedades Name, OpenOptions, DynamicQueueName y AlternateUserId del objeto MQQueue los valores proporcionados e intenta abrirlo.

Si no se abre correctamente, la llamada falla. Se genera un suceso de error para el objeto. Se establecen ReasonCode y CompletionCode, y MQSession ReasonCode y CompletionCode del objeto.

Los parámetros DynamicQueueName, QueueManagerName y AlternateUserId son opcionales y su valor predeterminado es "".

Si se deben leer las propiedades de la cola, también se debe especificar OpenOption MQOO_INQUIRE, además de las otras opciones.

No establezca QueueManagerName ni le asigne el valor "" si la cola que se va a abrir es local. De otro modo, puede asignarle el nombre del gestor de colas remoto que es el propietario de la cola y se realizará un intento de abrir una definición de local de la cola remota. Para obtener más información sobre la resolución de nombres de cola remota y el alias de gestor de colas, consulte ¿Qué son los alias? .

Si la propiedad Name se ha definido con un nombre de cola modelo, especifique el nombre de la cola dinámica que se va a crear en el parámetro DynamicQueueName$. Si el valor proporcionado en el parámetro DynamicQueueName$ es "", el valor definido en el objeto de cola y utilizado en la llamada de apertura es "AMQ.*". Consulte la sección “Creación de colas dinámicas” en la página 228 para obtener más información acerca de cómo asignar nombres a las colas dinámicas.

Definición

Definido en: Clase MQQueueManager.

Sintaxis

Sintaxis: set queue = MQQueueManager.**AccessQueue**(Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parámetros

Name$ Serie. Nombre de la cola de WebSphere MQ .

OpenOptions: Long. Las opciones que deben utilizarse cuando se abre la cola. Consulte la sección OpenOptions (MQLONG).

QueueManagerName$ Serie. El nombre del gestor de colas propietario de la cola que se va a abrir. El valor "" indica que el gestor de colas es local.

*DynamicQueueName$* Serie. El nombre asignado a la cola dinámica cuando se abre la cola y el parámetro Name$ especifica una cola modelo.

AlternateUserId$ Serie. El ID del usuario alternativo utilizado para validar el acceso cuando se abre la cola.

Método AddDistributionList

Crea un objeto MQDistributionList y establece su referencia de conexión al gestor de colas propietario.

Definida en:
clase MQQueueManager

Sintaxis: *set distributionlist* = **MQQueueManager.***AddDistributionList*

Método Backout

Restituye todas las transferencias y obtenciones de mensajes no confirmadas que se hayan producido como parte de una unidad de trabajo desde el último punto de sincronización.

Definido en: Clase MQQueueManager

Sintaxis: Llamar a *MQQueueManager.***Backout ()**

Método Begin

Inicia una unidad de trabajo coordinada por el gestor de colas. Las opciones de inicio afectan al comportamiento de este método.

Definida en:
clase MQQueueManager

Sintaxis: Llamada *MQQueueManager*.**Begin ()**

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQQueueManager y MQSession.

Definido en: Clase MQQueueManager

Sintaxis: Llamar a *MQQueueManager*.**ClearErrorCodes ()**

Método Commit

Confirma todas las colocaciones y obtenciones de mensaje que se han producido como parte de una unidad de trabajo desde el último punto de sincronización.

Definido en: Clase MQQueueManager

Sintaxis: Llamar a *MQQueueManager*.**Confirmar ()**

Método Connect

Conecta el objeto MQQueueManager a un gestor de colas real a través del cliente o servidor MQI de WebSphere MQ . Además de establecer la conexión, este método también abre el objeto de gestor de colas para que se pueda consultar.

Asigna a IsConnected el valor TRUE.

Se permite conectar con un gestor de colas un máximo de un objeto MQQueueManager por instancia de ActiveX.

Definido en: Clase MQQueueManager

Sintaxis: Llamar a *MQQueueManager*.**Conectar ()**

Método Disconnect

Desconecta el objeto MQQueueManager del gestor de colas.

Asigna a IsConnected el valor FALSE.

Todos los objetos Queue asociados al objeto MQQueueManager quedan inutilizables y no pueden volver a abrirse.

Todos los cambios sin confirmar (transferencias y obtenciones de mensajes) se confirman.

Definido en: Clase MQQueueManager

Sintaxis: Llamar a *MQQueueManager*.**Desconectar ()**

clase MQQueue

Esta clase representa el acceso a una cola WebSphere MQ . Esta conexión la proporciona un objeto MQQueueManager asociado. Cuando un objeto de esta clase se destruye, se cierra automáticamente.

Contención

La clase MQQueue está contenida en la clase MQQueueManager.

Creación

New crea un nuevo objeto MQQueue y devuelve a todas las propiedades sus valores iniciales. Como alternativa, se puede utilizar el método AccessQueue de la clase MQQueueManager.

Sintaxis

```
Dim que As New MQQueue Set que  = New MQQueue
```

Propiedades

- "Propiedad AlternateUserId" en la página 1077.

- “Propiedad RemoteQueueName” en la página 1085.
- “Propiedad ResolvedQueueManagerName” en la página 1085.
- “Propiedad ResolvedQueueName” en la página 1085.
- “Propiedad RetentionInterval” en la página 1085.
- “Propiedad Scope” en la página 1085.
- “Propiedad ServiceInterval” en la página 1085.
- “Propiedad ServiceIntervalEvent” en la página 1085.
- “Propiedad Shareability” en la página 1086.
- “Propiedad TransmissionQueueName” en la página 1086.
- “Propiedad TriggerControl” en la página 1086.
- “Propiedad TriggerData” en la página 1086.
- “Propiedad TriggerDepth” en la página 1086.
- “Propiedad TriggerMessagePriority” en la página 1087.
- “Propiedad TriggerType” en la página 1087.
- “Propiedad Usage” en la página 1087.

Métodos

- “Método ClearErrorCodes” en la página 1087
- “Método Close” en la página 1087
- “Método Get” en la página 1088
- “Método Open” en la página 1088
- “Método Put” en la página 1089

Acceso de propiedad

Si el objeto de cola no está conectado a un gestor de colas, podrá leer las siguientes propiedades:

- “Propiedad CompletionCode” en la página 1078
- “Propiedad OpenStatus” en la página 1083
- “propiedad ReasonCode” en la página 1084

y podrá leer y escribir en:

- “Propiedad AlternateUserId” en la página 1077
- “Propiedad CloseOptions” en la página 1078
- “propiedad ConnectionReference” en la página 1078
- “Propiedad Name” en la página 1082
- “Propiedad OpenOptions” en la página 1083

Si el objeto de cola está conectado a un gestor de colas, podrá leer todas las propiedades.

Propiedades de atributos de cola

Las propiedades no listadas en la sección anterior son todos los atributos de la cola subyacente de WebSphere MQ . Sólo puede accederse a ellas si el objeto está conectado a un gestor de colas y el ID de usuario tiene autorización para efectuar consultas o definiciones para dicha cola. Si se establece un ID de usuario alternativo y el ID de usuario actual está autorizado para utilizarlo, se comprueba la autorización del ID de usuario alternativo.

La propiedad debe ser una propiedad adecuada para el tipo de cola (QueueType) indicado. Consulte Atributos de colas para obtener más información.

Si estas condiciones no son aplicables, el acceso a la propiedad definirá un código de terminación (CompletionCode) MQCC_FAILED y uno de los siguientes códigos de razón (ReasonCodes):

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode es MQCC_WARNING)

Apertura de una cola

La única forma de crear un objeto MQQueue es utilizando el método MQQueueManager AccessQueue o mediante New. Un objeto MQQueue permanece abierto (OpenStatus=TRUE) hasta que se cierra o se suprime o hasta que se suprime el objeto de creación de gestor de colas o se pierde la conexión con el gestor de colas. El valor de la propiedad MQQueue CloseOptions controla el comportamiento de la operación de cierre que se produce al suprimir el objeto MQQueue.

El método MQQueueManager AccessQueue abre la cola utilizando el parámetro OpenOptions. El método MQQueue.Open abre la cola utilizando la propiedad OpenOptions. WebSphere MQ valida OpenOptions con la autorización de usuario como parte del proceso de cola abierta.

Propiedad AlternateUserId

Lectura-grabación. ID de usuario alternativo utilizado para validar el acceso a una cola cuando se abre.

Esta propiedad no se puede establecer mientras el objeto está abierto (es decir, cuando IsOpen es TRUE).

Definido en: clase MQQueue

Tipo de datos: String de 12 caracteres

Sintaxis: Para obtener: *altuser $= MQQueue*.**AlternateUserId**

Para establecer: *MQQueue.***AlternateUserId** = *altuser $*

Propiedad BackoutRequeueName

Solo lectura. Es el atributo BackOutRequeueQName de la MQI.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *backoutrequeuename $= MQQueue*.**BackoutRequeueNombre**

Propiedad BackoutThreshold

Solo lectura. Atributo BackoutThreshold de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- Consulte BackoutThreshold (MQLONG).

Sintaxis: Para obtener: *backoutthreshold & = MQQueue.***BackoutThreshold**

Propiedad BaseQueueName

Solo lectura. Es el nombre de la cola a la que resuelve el alias.

Solo es válido para colas alias.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *baseqname $= MQQueue***.BaseQueueNombre**

Propiedad CloseOptions

Lectura-Escritura. Opciones utilizadas para controlar lo que sucede cuando se cierra la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE y MQCO_DELETE_PURGE solo son válidos para colas dinámicas.

Sintaxis: Para obtener: *closeopt & = MQQueue***.CloseOptions**

Para establecer: *MQQueue***.CloseOptions** = *closeopt &*

Propiedad CompletionCode

Solo lectura. Devuelve el código de terminación establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = MQQueue***.CompletionCode**

propiedad ConnectionReference

Lectura-grabación. Define el objeto del gestor de colas al que pertenece un objeto de cola. La referencia de conexión no se puede escribir mientras una cola está abierta.

Definido en: clase MQQueue

Tipo de datos: MQQueueManager

Valores:

- Una referencia a un objeto de gestor de colas de WebSphere MQ activo

Sintaxis: Para establecer: *set MQQueue***.ConnectionReference** = *ConnectionReference*

Para obtener: *set ConnectionReference = MQQueue***.ConnectionReference**

Propiedad CreationDateTime

Solo lectura. Es la fecha y la hora de creación de la cola.

Definido en: clase MQQueue

Tipo de datos: Variant de tipo 7 (date/time) o EMPTY

Sintaxis: Para obtener: *datetime = MQQueue***.CreationDateTime**

Propiedad CurrentDepth

Solo lectura. Número de mensajes que hay en la cola actualmente.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *currentdepth & = MQQueue*.**CurrentDepth**

Propiedad DefaultInputOpenOption

Solo lectura. Controla la forma en que la cola se abre si OpenOptions especifica MQOO_INPUT_AS_Q_DEF.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

Sintaxis: Para obtener: *defaultinop & = MQQueue*.**DefaultInputOpenOption**

Propiedad DefaultPersistence

Solo lectura. Es la persistencia predeterminada de los mensajes en una cola.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *defpersistence & = MQQueue*.**DefaultPersistence**

Propiedad DefaultPriority

Solo lectura. Es la prioridad predeterminada de los mensajes en una cola.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *defpriority & = MQQueue*.**DefaultPriority**

Propiedad DefinitionType

Solo lectura. El tipo de definición de la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

Sintaxis: Para obtener: *deftype & = MQQueue*.**DefinitionType**

Propiedad DepthHighEvent

Solo lectura. Es el atributo QDepthHighEvent de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *depthhighevent & = MQQueue.***DepthHighEvent**

Propiedad DepthHighLimit

Solo lectura. Es el atributo QDepthHighLimit de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *depthhighlimit & = MQQueue.***DepthHighLimit**

Propiedad DepthLowEvent

Solo lectura. Es el atributo QDepthLowEvent de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *depthlowevent & = MQQueue.***DepthLowEvent**

Propiedad DepthLowLimit

Solo lectura. Es el atributo QDepthLowLimit de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *depthlowlimit & = MQQueue.***DepthLowLimit**

Propiedad DepthMaximumEvent

Solo lectura. Es el atributo QDepthMaxEvent de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxis: Para obtener: *depthmaximumevent & = MQQueue.***DepthMaximumEvent**

Propiedad Description

Solo lectura. Es una descripción de la cola.

Definido en: clase MQQueue

Tipo de datos: String de 64 caracteres

Sintaxis: Para obtener: *description $= MQQueue***.Descripción**

Propiedad DynamicQueueName

Lectura-escritura, solo lectura cuando la cola está abierta.

Controla el nombre de la cola dinámica que se usa cuando se abre una cola modelo. El usuario lo puede definir con un comodín como un conjunto de propiedades (solo cuando la cola está cerrada) o como un parámetro de MQQueueManager.AccessQueue().

El nombre real de la cola dinámica se obtiene consultando QueueName.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Valores:

- Cualquier nombre de cola válido de WebSphere MQ .

Sintaxis: Para establecer: *MQQueue***.DynamicQueueName** = *dynamicqueuename $*

Para obtener: *dynamicqueuename $ = MQQueue***.DynamicQueueNombre**

Propiedad HardenGetBackout

Solo lectura. Indica si va a mantenerse un contador de restituciones exacto.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

Sintaxis: Para obtener: *hardengetback & = MQQueue***.HardenGetBackout**

Propiedad InhibitGet

Lectura-grabación. E atributo InhibitGet de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

Sintaxis: Para obtener: *getstatus & = MQQueue***.InhibitGet**

Para establecer: *MQQueue***.InhibitGet** = *getstatus &*

Propiedad InhibitPut

Lectura-grabación. Es el atributo InhibitPut de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

Sintaxis: Para obtener: *putstatus & = MQQueue***.InhibitPut**

Para establecer: *MQQueue***.InhibitPut** = *putstatus &*

Propiedad InitiationQueueName

Solo lectura. Es el nombre de la cola de inicio.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *initqname $= MQQueue*.**InitiationQueueNombre**

propiedad IsOpen

Indica si la cola está abierta.

Solo lectura.

Definido en: clase MQQueue

Tipo de datos: Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *open = MQQueue*.**IsOpen**

Propiedad MaximumDepth

Solo lectura. Es la profundidad máxima de la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *maxdepth & = MQQueue*.**MaximumDepth**

Propiedad MaximumMessageLength

Solo lectura. Es la longitud máxima de mensaje permitida en esta cola en bytes.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *maxmlength & = MQQueue*.**MaximumMessageLength**

Propiedad MessageDeliverySequence

Solo lectura. Secuencia de entrega de mensajes.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQMDS_PRIORITY
- MQMDS_FIFO

Sintaxis: Para obtener: *messdelseq & = MQQueue*.**MessageDeliverySequence**

Propiedad Name

Lectura-grabación. Atributo Queue de la MQI. Esta propiedad no se puede escribir una vez abierta la MQQueue.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *name $= MQQueue*.**name**

Para establecer: *MQQueue*.**name** = *nombre $*

Nota: Visual Basic reserva la propiedad "Name" para su uso en la interfaz visual. Por lo tanto, cuando lo use en Visual Basic, emplee minúsculas, es decir, "name".

Propiedad ObjectHandle

Solo lectura. El descriptor de contexto de objeto para el objeto de cola WebSphere MQ .

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *hobj & = MQQueue.***ObjectHandle**

Propiedad OpenInputCount

Solo lectura. Es el número de aperturas para entrada.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *openincount & = MQQueue***.OpenInputCount**

Propiedad OpenOptions

Lectura-grabación. Opciones que se usan para abrir la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- Consulte la sección OpenOptions (MQLONG).

Sintaxis: Para obtener: *openopt & = MQQueue***.OpenOptions**

Para establecer: *MQQueue.***OpenOptions** = *openopt &*

Propiedad OpenOutputCount

Solo lectura. Es el número de aperturas para salida.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *openoutcount & = MQQueue***.OpenOutputCount**

Propiedad OpenStatus

Solo lectura. Indica si la cola está o no abierta. El valor inicial es TRUE después del método AccessQueue o FALSE después de New.

Definido en: clase MQQueue

Tipo de datos: Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *status & = MQQueue***.OpenStatus**

Propiedad ProcessName

Solo lectura. Es el atributo ProcessName de la MQI.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *procname $ =MQQueue***.ProcessName**

Propiedad QueueManagerName

Lectura-grabación. El nombre del gestor de colas de WebSphere MQ .

Definido en: clase MQQueue

Tipo de datos: String

Sintaxis: Para obtener: *QueueManagerName$ = MQQueue***.QueueManagerName**

Para establecer: *MQQueue***.QueueManagerName** = *QueueManagerName$*

Propiedad QueueType

Solo lectura. Es el atributo QType de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

Sintaxis: Para obtener: *queuetype & = MQQueue***.QueueType**

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = MQQueue***.ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido en: clase MQQueue

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQQueue***.ReasonName**

Propiedad RemoteQueueManagerName

Solo lectura. Nombre del gestor de colas remoto. Solo es válido para colas remotas.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *remqmanname $= MQQueue*.**RemoteQueueManagerName**

Propiedad RemoteQueueName

Solo lectura. Es el nombre de la cola tal como se conoce en el gestor de colas remoto. Solo es válido para colas remotas.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *remqname $= MQQueue*.**RemoteQueueName**

Propiedad ResolvedQueueManagerName

Solo lectura. Nombre del gestor de colas de destino final tal como lo conoce el gestor de colas local.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *resqmanname $= MQQueue*.**ResolvedQueueManagerName**

Propiedad ResolvedQueueName

Solo lectura. Nombre de la cola de destino final tal como lo conoce el gestor de colas local.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *resqname $= MQQueue*.**ResolvedQueueNombre**

Propiedad RetentionInterval

Solo lectura. Es el período de tiempo durante el cual hay que retener la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *retinterval & = MQQueue*.**RetentionInterval**

Propiedad Scope

Solo lectura. Controla si también existe una entrada para esta cola en un directorio de celdas.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQSCO_Q_MGR
- MQSCO_CELL

Sintaxis: Para obtener: *scope & = MQQueue*.**Scope**

Propiedad ServiceInterval

Solo lectura. Es el atributo QServiceInterval de la MQI.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *serviceinterval & = MQQueue.***ServiceInterval**

Propiedad ServiceIntervalEvent

Solo lectura. El atributo MQI QServiceIntervalEvent.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

Sintaxis: Para obtener: *serviceintervalevent & = MQQueue.***ServiceIntervalEvent**

Propiedad Shareability

Solo lectura. Indica la posibilidad de compartir la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

Sintaxis: Para obtener: *shareability & = MQQueue***.Shareability**

Propiedad TransmissionQueueName

Solo lectura. Nombre de la cola de transmisión. Solo es válido para colas remotas.

Definido en: clase MQQueue

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *transqname $= MQQueue***.TransmissionQueueName**

Propiedad TriggerControl

Lectura-grabación. Control de desencadenante.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQTC_OFF
- MQTC_ON

Sintaxis: Para obtener: *trigcontrol & = MQQueue***.TriggerControl**

Para establecer: *MQQueue***.TriggerControl** = *trigcontrol &*

Propiedad TriggerData

Lectura-grabación. Datos del desencadenante.

Definido en: clase MQQueue

Tipo de datos: String de 64 caracteres

Sintaxis: Para obtener: *trigdata $= MQQueue***.TriggerData**

Para establecer: *MQQueue***.TriggerData** = *trigdata $*

Propiedad TriggerDepth

Lectura-grabación. El número de mensajes que tienen que estar en la cola antes de que se escriba un mensaje desencadenante.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *trigdepth & = MQQueue***.TriggerDepth**

Para establecer: *MQQueue***.TriggerDepth** = *trigdepth &*

Propiedad TriggerMessagePriority

Lectura-grabación. Umbral de prioridad del mensaje para activaciones.

Definido en: clase MQQueue

Tipo de datos: Long

Sintaxis: Para obtener: *trigmesspriority & = MQQueue***.TriggerMessagePriority**

Para establecer: *MQQueue***.TriggerMessagePriority** = *trigmesspriority &*

Propiedad TriggerType

Lectura-grabación. Tipo de desencadenante.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

Sintaxis: Para obtener: *trigtype & = MQQueue***.TriggerType**

Para establecer: *MQQueue***.TriggerType** = *Trigtype &*

Propiedad Usage

Solo lectura. Indica la finalidad de la cola.

Definido en: clase MQQueue

Tipo de datos: Long

Valores:

- MQUS_NORMAL
- MQUS_TRANSMISSION

Sintaxis: Para obtener: *usage & = MQQueue***.Uso**

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQQueue y MQSession.

Definido en: clase MQQueue

Sintaxis: Llamar a *MQQueue***.ClearErrorCodes ()**

Método Close

Cierra una cola utilizando los valores actuales de CloseOptions.

Definido en: clase MQQueue

Sintaxis: Llame a *MQQueue*.**Cerrar ()**

Método Get

Recupera un mensaje de la cola.

Este método toma un objeto MQMessage como un parámetro, utilizando algunos de los campos en el MQMD del objeto como parámetros de entrada. Concretamente, se utilizan los campos MessageId y CorrelId, por lo que es importante asegurarse de que estos campos se establecen según sea necesario. Para obtener más información sobre estos campos, consulte MsgId (MQBYTE24) y CorrelId (MQBYTE24) .

Si el método no se ejecuta correctamente, el objeto MQMessage no sufre cambios. Si es correcto, las partes MQMD y Message Data del objeto MQMessage se sustituyen por el Message Data y MQMD y del mensaje de entrada. Las propiedades de control de MQMessage se definen como sigue

- **MessageLength** se establece en la longitud del mensaje WebSphere MQ
- **DataLength** se establece en la longitud del mensaje WebSphere MQ
- **DataOffset** se establece en cero

Definida en:
clase MQQueue

Sintaxis: Llame a *MQQueue*.**Obtener**(*Message, GetMsgOptions, GetMsgLength*)

Parámetros
Mensaje:

El objeto de MQMessage que representa el mensaje que debe recuperarse.

GetMsgOptions:

El objeto MQGetMessageOptions opcional para controlar la operación get. Si no se especifica este parámetro, se utilizan las MQGetMessageOptions de forma predeterminada.

GetMsgLength:

Valor de longitud de 2 o 4 bytes opcional para controlar la longitud máxima del mensaje de WebSphere MQ que se recupera de la cola.

Si se especifica la opción MQGMO_ACCEPT_TRUNCATED_MSG, la operación GET tiene éxito con un código de terminación de MQCC_WARNING y un código de razón de MQRC_TRUNCATED_MSG_ACCEPTED si el tamaño del mensaje supera la longitud especificada.

MessageData contiene los primeros bytes de datos de GetMsgLength.

Si ***no*** se especifica MQGMO_ACCEPT_TRUNCATED_MSG, y el tamaño del mensaje supera la longitud especificada, se devuelve el código de terminación de MQCC_FAILED junto con el código de razón MQRC_TRUNCATED_MESSAGE_FAILED.

Si el contenido del almacenamiento intermedio de mensajes es indefinido, la longitud total del mensaje se establece en la longitud completa del mensaje que se hubiera recuperado.

Si el parámetro de longitud del mensaje no se ha especificado, la longitud del almacenamiento intermedio de mensajes se ajusta automáticamente para adoptar como mínimo el tamaño del mensaje entrante.

Método Open

Abre una cola utilizando los valores actuales de:

1. QueueName
2. QueueManagerName
3. AlternateUserId
4. DynamicQueueName

Definida en:
clase MQQueue

Sintaxis: Llame a *MQQueue*.**Abrir ()**

Método Put

Coloca un mensaje en la cola.

Este método utiliza un objeto MQMessage como parámetro. Las propiedades de descriptor de mensaje (MQMD) de este objeto se pueden alterar como resultado de este método. Los valores que tienen inmediatamente después de que se haya ejecutado este método son los valores que se han colocado en WebSphere MQ.

Las modificaciones en el objeto MQMessage después de que se haya completado la colocación no afectan al mensaje real en la cola de WebSphere MQ .

Definida en:
clase MQQueue

Sintaxis: Llame a *MQQueue*.**Put**(*Mensaje, Opciones de PutMsg)*

Parámetros
Mensaje

Objeto MQMessage que representa el mensaje que se va a colocar.

PutMsgOptions

Objeto MQPutMessageOptions que contiene opciones para controlar la operación de transferencia. Si no se especifican, se utilizan las PutMessageOptions predeterminadas.

Clase MQMessage

Esta clase representa un mensaje WebSphere MQ . Incluye propiedades para encapsular el descriptor de mensaje WebSphere MQ (MQMD) y proporciona un almacenamiento intermedio para contener los datos de mensaje definidos por la aplicación.

La clase incluye métodos de escritura para copiar los datos de una aplicación ActiveX en un objeto de MQMessage. Del mismo modo, la clase incluye métodos de escritura para copiar los datos de un objeto MQMessage en una aplicación ActiveX. La clase gestiona la asignación y desasignación automática de memoria para el almacenamiento intermedio. La aplicación no tiene que declarar el tamaño del almacenamiento intermedio cuando se crea un objeto MQMessage, puesto que el almacenamiento intermedio crece para que puedan caber los datos que se graban en el mismo.

No puede colocar un mensaje en una cola de WebSphere MQ si el tamaño del almacenamiento intermedio supera la propiedad de longitud MaximumMessagede dicha cola.

Una vez construido, un objeto MQMessage puede colocarse en una cola de WebSphere MQ utilizando el método MQQueue.Put . Este método realiza una copia del MQMD y de las partes de datos del mensaje del objeto y coloca dicha copia en la cola. Por lo tanto, la aplicación puede modificar o suprimir un objeto MQMessage después de la transferencia, sin afectar al mensaje en la cola de WebSphere MQ . El gestor de colas puede ajustar algunos de los campos del MQMD cuando copia el mensaje en la cola WebSphere MQ .

Se puede leer un mensaje en un objeto MQMessage utilizando el método MQQueue.Get. Esto sustituye cualquier MQMD o datos del mensaje que ya puedan haber estado en el objeto MQMessage por los valores del mensaje entrante. Ajusta el tamaño del almacenamiento intermedio del objeto MQMessage para que coincida con el tamaño de los datos del mensaje de entrada.

Contención

Los mensajes los contiene la clase MQSession.

Creación

New crea un objeto MQMessage. Las propiedades de su descriptor de mensaje adoptan inicialmente los valores predeterminados y su almacenamiento intermedio de datos del mensaje está vacío.

Sintaxis

```
Dim msg As New MQMessage or Set msg = New MQMessage
```

Propiedades

Las propiedades de control son las siguientes:

- "Propiedad CompletionCode" en la página 1092
- "Propiedad DataLength" en la página 1092
- "Propiedad DataOffset" en la página 1093
- "Propiedad MessageLength" en la página 1093
- "propiedad ReasonCode" en la página 1093
- "propiedad ReasonName" en la página 1093

Las propiedades del descriptor de mensaje son las siguientes:

- "propiedad AccountingToken" en la página 1093
- "Propiedad AccountingTokenHex" en la página 1094
- "Propiedad ApplicationIdData" en la página 1094
- "Propiedad ApplicationOriginData" en la página 1094
- "Propiedad BackoutCount" en la página 1094
- "propiedad CharacterSet" en la página 1095
- "Propiedad CorrelationId" en la página 1095
- "Propiedad CorrelationIdHex" en la página 1095
- "La propiedad de codificación" en la página 1096
- "Propiedad Expiry" en la página 1096
- "propiedad Feedback" en la página 1097
- "Propiedad Format" en la página 1097
- "propiedad GroupId" en la página 1097
- "Propiedad GroupIdHex" en la página 1097
- "Propiedad MessageData" en la página 1098
- "propiedad MessageFlags" en la página 1098
- "propiedad MessageId" en la página 1098
- "Propiedad MessageIdHex" en la página 1098
- "propiedad MessageSequenceNumber" en la página 1099
- "Propiedad MessageType" en la página 1099
- "propiedad Offset" en la página 1099
- "Propiedad OriginalLength" en la página 1100
- "propiedad Persistencia" en la página 1100
- "Propiedad Priority" en la página 1100
- "Propiedad PutApplicationName" en la página 1100
- "Propiedad PutApplicationType" en la página 1100

- “Propiedad PutDateTime” en la página 1101
- “Propiedad ReplyToQueueManagerName” en la página 1101
- “Propiedad ReplyToQueueName” en la página 1101
- “Propiedad Report” en la página 1101
- “Propiedad TotalMessageLength” en la página 1102
- “Propiedad UserId” en la página 1102

Métodos

- “Método ClearErrorCodes” en la página 1102
- “Método ClearMessage” en la página 1102
- “Método Read” en la página 1102
- “Método ReadBoolean” en la página 1102
- “Método ReadByte” en la página 1103
- “Método ReadDecimal2” en la página 1103
- “Método ReadDecimal4” en la página 1103
- “Método ReadDouble” en la página 1103
- “Método ReadDouble4” en la página 1103
- “Método ReadFloat” en la página 1104
- “Método ReadInt2” en la página 1104
- “Método ReadInt4” en la página 1104
- “Método ReadLong” en la página 1104
- “Método ReadNullTerminatedString” en la página 1104
- “Método ReadShort” en la página 1105
- “método ReadString” en la página 1105
- “Método ReadUInt2” en la página 1105
- “Método ReadUnsignedByte” en la página 1105
- “Método ReadUTF” en la página 1106
- “Método ResizeBuffer” en la página 1106
- “Método Write” en la página 1106
- “Método WriteBoolean” en la página 1107
- “Método WriteByte” en la página 1107
- “Método WriteDecimal2” en la página 1107
- “Método WriteDecimal4” en la página 1107
- “Método WriteDouble” en la página 1107
- “Método WriteDouble4” en la página 1108
- “Método WriteFloat” en la página 1108
- “Método WriteInt2” en la página 1108
- “Método WriteInt4” en la página 1108
- “Método WriteLong” en la página 1109
- “Método WriteNullTerminatedString” en la página 1109
- “Método WriteShort” en la página 1109
- “Método WriteString” en la página 1109
- “Método WriteUInt2” en la página 1110

- “Método WriteUnsignedByte” en la página 1110
- “Método WriteUTF” en la página 1110

Acceso a propiedades

Todas las propiedades pueden leerse en cualquier momento.

Las propiedades de control son de solo lectura, excepto DataOffset que es de lectura/escritura.
Las propiedades del descriptor de mensaje son todas de lectura/escritura, excepto BackoutCount y TotalMessageLength que son ambas de solo lectura.

Sin embargo, tenga en cuenta que algunas de las propiedades MQMD pueden ser modificadas por el gestor de colas cuando el mensaje se coloca en una cola WebSphere MQ . Consulte los campos de MQMD para obtener detalles acerca de cómo se pueden modificar.

Conversión de datos

Puede pasar datos binarios a un mensaje de WebSphere MQ estableciendo la propiedad CharacterSet en el Identificador de juego de caracteres codificado del gestor de colas (MQCCSI_Q_MGR) y pasándole una serie. Puede utilizar la función chr $para establecer datos que no sean de caracteres en la serie.

Los métodos de lectura y escritura realizan la conversión de los datos. Se convierten entre los formatos internos ActiveX y los formatos de mensaje WebSphere MQ tal como los definen las propiedades Encoding y CharacterSet del descriptor de mensaje. Cuando escriba un mensaje, establezca los valores de Encoding y CharacterSet de modo que coincidan con las características del destinatario del mensaje antes de emitir un método de escritura. Normalmente cuando lee un mensaje este paso no es necesario, ya que estos valores se habrán establecido a partir del MQMD de entrada.

Este es un paso de conversión de datos adicional que se produce después de cualquier conversión realizada mediante el método MQQueue.Get.

Propiedad CompletionCode

Solo lectura. Devuelve el código de finalización WebSphere MQ establecido por el método o acceso de propiedad más reciente emitido para este objeto.

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = MQMessage*.**CompletionCode**

Propiedad DataLength

Solo lectura. Esta propiedad devuelve el valor:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Se puede utilizar antes de un método Read para comprobar que el número esperado de caracteres está realmente en el búfer.

El valor inicial es cero.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *bytesleft & = MQMessage*.**DataLength**

Propiedad DataOffset

Lectura-grabación. Posición actual del objeto del mensaje dentro de la parte de datos del mensaje.

El valor se expresa como el desplazamiento en bytes desde el inicio del búfer de datos del mensaje; el primer carácter del búfer se corresponde con un valor de DataOffset de cero.

Un método de lectura o escritura inicia su operación en el carácter referenciado por DataOffset. Estos métodos procesan secuencialmente los datos del búfer a partir de esta posición y actualizan DataOffset para que apunte al byte (si lo hay) inmediatamente posterior al último byte procesado.

DataOffset solo puede tomar valores en el rango que va de cero a MessageLength inclusive. Cuando DataOffset = MessageLength, está apuntando al final, que es el primer carácter no válido del búfer. Los métodos de escritura están permitidos en esta situación: extienden los datos del búfer e incrementan MessageLength en el número de bytes añadidos. No se puede leer más allá del final del búfer.

El valor inicial es cero.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *currpos & = MQMessage*.**DataOffset**

Para establecer: *MQMessage*.**DataOffset** = *currpos &*

Propiedad MessageLength

Solo lectura. Devuelve, en caracteres, la longitud total de la porción de datos del objeto de mensaje, independientemente del valor de DataOffset.

El valor inicial es cero. Se establece a la longitud del mensaje entrante después de una invocación del método Get que ha referenciado a este objeto de mensaje. Se incrementa si la aplicación utiliza el método Write para añadir datos al objeto. No se ve afectado por los métodos Read.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *msglength & = MQMessage*.**MessageLength**

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el método o acceso de propiedad más recientes emitidos contra este objeto.

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = MQMessage*.**ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE". **Definido en:** clase MQMessage

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQMessage*.**ReasonName**

propiedad AccountingToken

Lectura-grabación. El AccountingToken del MQMD - parte del contexto de identidad del mensaje.

Su valor inicial es todo nulos.

Definido en: clase MQMessage

Tipo de datos: String de 32 caracteres

Sintaxis: Para obtener: *actoken $= MQMessage*.**AccountingToken**

Para establecer: *MQMessage.***AccountingToken** = *actoken $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener más información sobre cuándo hay que utilizar AccountingTokenHex en lugar de la propiedad AccountingToken.

Propiedad AccountingTokenHex

Lectura-grabación. El AccountingToken del MQMD - parte del contexto de identidad del mensaje.

Cada dos caracteres representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 64 caracteres hexadecimales válidos.

Su valor inicial es "0...0"

Definido en: clase MQMessage

Tipo de datos: String de 64 caracteres hexadecimales que representan 32 caracteres ASCII.

Sintaxis: Para obtener: *actokenh $= MQMessage*.**AccountingTokenHex**

Para establecer: *MQMessage.***AccountingTokenHex** = *actokenh $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener más información sobre cuándo hay que utilizar AccountingTokenHex en lugar de la propiedad AccountingToken.

Propiedad ApplicationIdData

Lectura-grabación. El ApplIdentityData del MQMD. Forma parte del contexto de identidad del mensaje.

Su valor inicial contiene únicamente blancos.

Definido en: clase MQMessage

Tipo de datos: String de 32 caracteres

Sintaxis: Para obtener: *applid $= MQMessage*.**ApplicationIdDatos**

Para establecer: *MQMessage*.**ApplicationIdData** = *applid $*

Propiedad ApplicationOriginData

Lectura-grabación. ApplOriginData del MQMD. Forma parte del contexto de origen del mensaje.

Su valor inicial contiene únicamente blancos.

Definido en: clase MQMessage

Tipo de datos: String de 4 caracteres

Sintaxis: Para obtener: *applor $= MQMessage*.**ApplicationOriginData**

Para establecer: *MQMessage*.**ApplicationOriginData** = *applor $*

Propiedad BackoutCount

Solo lectura. Es el BackoutCount (recuento de restiticiones) del MQMD.

Su valor inicial es 0

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *backoutct & = MQMessage***.BackoutCount**

propiedad CharacterSet

Lectura-grabación. El MQMD CodedCharSetId.

Su valor inicial es el valor especial **MQCCSI_Q_MGR** .

Si CharacterSet se establece en **MQCCSI_Q_MGR**, la página de códigos del entorno local actual se utiliza para la conversión de caracteres en el método WriteString . En las aplicaciones de servidor, se utiliza la página de códigos del gestor de colas. En las aplicaciones de cliente, es la página de códigos predeterminada del entorno local actual.

Por ejemplo:

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

donde 'n', que es mayor o igual que cero y menor o igual que 255, tiene como resultado la escritura de un solo byte con el valor 'n' en el almacenamiento intermedio.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *:30ccid& = MQMessage* **.CharacterSet**

Para establecer: *MQMessage***.CharacterSet**= *ccid &*

Ejemplo

Si desea que la serie de caracteres se escriba con la página de códigos 437, indique lo siguiente:

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Establezca el valor que desee para CharacterSet antes de emitir llamadas WriteString.

Propiedad CorrelationId

Lectura-grabación. El CorrelationId que hay que incluir en el MQMD de un mensaje cuando se coloca en una cola. Además, el ID con el que debe coincidir cuando se obtiene un mensaje de una cola.

Su valor inicial es nulo.

Definido en: clase MQMessage

Tipo de datos: String de 24 caracteres

Sintaxis: Para obtener: *correlid $= MQMessage***.CorrelationId** Para establecer: *MQMessage***.CorrelationId** = *correlid $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener más información sobre cuándo hay que utilizar CorrelationIdHex en lugar de la propiedad CorrelationId.

Propiedad CorrelationIdHex

Lectura-grabación. El CorrelationId que hay que incluir en el MQMD de un mensaje cuando se coloca en una cola. Además, el CorrelationId con el que se compara al obtener un mensaje de una cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0...0".

Definido en: clase MQMessage

Tipo de datos: String de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *correlidh $= MQMessage*.**CorrelationIdHex**

Para establecer: *MQMessage*.**CorrelationIdHex** = *correlidh $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener una explicación de cuándo usar correlationIdHex en lugar de la propiedad CorrelationId.

La propiedad de codificación

Lectura-grabación. Campo del MQMD que identifica la representación utilizada en los valores numéricos de los datos de mensaje de aplicación.

Su valor inicial es el valor especial MQENC_NATIVE, que varía según la plataforma.

Esta propiedad la usan los métodos siguientes:

- Método ReadDecimal2
- Método ReadDecimal4
- Método ReadDouble
- Método ReadDouble4
- Método ReadFloat
- Método ReadInt2
- Método ReadInt4
- Método ReadLong
- Método ReadShort
- Método ReadUInt2
- Método WriteDecimal2
- Método WriteDecimal4
- Método WriteDouble
- Método WriteDouble4
- Método WriteFloat
- Método WriteInt2
- Método WriteInt4
- Método WriteLong
- Método WriteShort
- Método WriteUInt2

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *encoding & = MQMessage*.**Codificación** Para establecer: *MQMessage*.**Codificación** = *codificación &*

Si está preparando la escritura de datos en el búfer de mensajes, tendrá que definir este campo para que coincida con las características de la plataforma del gestor de colas receptor si este no es capaz de realizar su propia conversión de datos.

Propiedad Expiry

Lectura-grabación. Campo de tiempo de caducidad del MQMD, en décimas de segundo.

Su valor inicial es el valor especial MQEI_UNLIMITED

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *expire & = MQMessage*.**Caducidad**

Para establecer: *MQMessage*.**Caducidad** = *caducidad &*

propiedad Feedback

Lectura-grabación. Campo feedback de MQMD.

Su valor inicial es el valor especial MQFB_NONE.

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- Consulte Feedback.

Sintaxis: Para obtener: *feedback & = MQMessage*.**Comentarios**

Para establecer: *MQMessage*.**Comentarios** = *comentarios &*

Propiedad Format

Lectura-grabación. Campo MsgType del MQMD. Proporciona el nombre de un formato incorporado o definido por el usuario que describe la naturaleza de los datos del mensaje.

Su valor inicial es el valor especial MQFMT_NONE.

Definido en: clase MQMessage

Tipo de datos: String de 8 caracteres

Sintaxis: Para obtener: *format $= MQMessage*.**Formato**

Para establecer: *MQMessage*.**Formato** = *formato $*

propiedad GroupId

Lectura-grabación. El GroupId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola. Además, el ID con el que debe coincidir cuando se obtiene un mensaje de una cola. Su valor inicial es todo nulos.

Definida en:
Clase MQMessage

Tipo de datos:
Serie de 24 caracteres

Sintaxis: Para obtener: *groupid $= MQMessage*.**GroupId**

Para establecer: *MQMessage*.**GroupId** = *groupid $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener más información acerca de cuándo debe utilizar GroupIdHex, en lugar de la propiedad GroupId.

Propiedad GroupIdHex

Lectura-grabación. El GroupId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola. Además, el ID con el que debe coincidir cuando se obtiene un mensaje de una cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0...0".

Definida en:
Clase MQMessage

Tipo de datos:
Cadena de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *groupidh $= MQMessage*.**GroupIdHex**

Para establecer: *MQMessage*.**GroupIdHex** = *groupidh $*

Consulte "Propiedades del descriptor de mensaje" en la página 1054 para obtener más información acerca de cuándo debe utilizar GroupIdHex, en lugar de la propiedad GroupId.

Propiedad MessageData

Lectura-grabación. Recupera o define todo el contenido de un mensaje como una cadena de caracteres.

Definido en: clase MQMessage

Tipo de datos: Variant

Nota: El tipo de datos utilizado por esta propiedad es Variant, pero MQAX espera que este sea un tipo de variante de String. Si se pasa una variante distinta de este tipo, se devolverá el error MQRC_OBJECT_TYPE_ERROR.

Sintaxis: Para obtener: *String$* = *MQMessage***.MessageData**

Para establecer: *MQMessage***.MessageData** = *String$*

propiedad MessageFlags

Lectura-Escritura. Indicadores de mensajes que especifican información de control de segmentación. El valor inicial es 0.

Definida en:
Clase MQMessage

Tipo de datos:
Long

Valores:
Consulte MsgFlags (MQLONG).

Sintaxis: Para obtener: *messageflags & = MQMessage*.**MessageFlags**

Para establecer: *MQMessage*.**MessageFlags** = *messageflags &*

propiedad MessageId

Lectura-grabación. El MessageId que hay que incluir en el MQMD de un mensaje cuando se coloca en una cola. Además, el ID con el que debe coincidir cuando se obtiene un mensaje de una cola.

Su valor inicial es todo nulos.

Definido en: clase MQMessage

Tipo de datos: String de 24 caracteres

Sintaxis: Para obtener: *messageid $= MQMessage***.MessageId**

Para establecer: *MQMessage***.MessageId** = *messageid $*

Consulte "Propiedades del descriptor de mensaje" en la página 1054 para obtener más información sobre cuándo hay que utilizar MessageIdHex en lugar de la propiedad MessageId.

Propiedad MessageIdHex

Lectura-grabación. El MessageId que hay que incluir en el MQMD de un mensaje cuando se coloca en una cola. Además, el MessageId con el que se compara al obtener un mensaje de una cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0...0".

Definido en: clase MQMessage

Tipo de datos: String de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *messageidh $= MQMessage*.**MessageIdHex**

Para establecer: *MQMessage*.**MessageIdHex** = *messageidh $*

Consulte “Propiedades del descriptor de mensaje” en la página 1054 para obtener más información sobre cuándo hay que utilizar MessageIdHex en lugar de la propiedad MessageId.

propiedad MessageSequenceNumber

Lectura-Escritura. Información de secuencia que identifica un mensaje dentro de un grupo. El valor inicial es 1.

Definida en:
Clase MQMessage

Tipo de datos:
Long

Valores:
Consulte MsgSeqNumber (MQLONG).

Sintaxis: Para obtener: *sequencenumber & = MQMessage*.**SequenceNumber**

Para establecer: *MQMessage*.**SequenceNumber** = *sequencenumber &*

Propiedad MessageType

Lectura-grabación. Campo MsgType del MQMD.

Su valor inicial es MQMT_DATAGRAM.

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- Consulte MsgType (MQLONG).

Sintaxis: Para obtener: *msgtype & = MQMessage*.**MessageType**

Para establecer: *MQMessage*.**MessageType** = *msgtype &*

propiedad Offset

Lectura-Escritura. El desplazamiento en un mensaje segmentado. El valor inicial es 0.

Definida en:
Clase MQMessage

Tipo de datos:
Long

Valores:
Consulte Offset (MQLONG).

Sintaxis: Para obtener: *offset & = MQMessage*.**Offset**

Para establecer: *MQMessage*.**Offset** = *offset &*

Propiedad OriginalLength

Lectura-Escritura. La longitud original de un mensaje segmentado. El valor inicial es MQOL_UNDEFINED.

Definida en:
Clase MQMessage

Tipo de datos:
Long

Valores:
Consulte OriginalLength (MQLONG).

Sintaxis: Para obtener: *originallength & = MQMessage*.**OriginalLength**

Para establecer: *MQMessage*.**OriginalLength** = *originallength &*

propiedad Persistencia

Lectura-grabación. El valor de persistencia del mensaje.

Su valor inicial es MQPER_PERSISTENCE_AS_Q_DEF.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *persist & = MQMessage*.**Persistencia**

Para establecer: *MQMessage*.**Persistencia** = *persist &*

Propiedad Priority

Lectura-grabación. La prioridad del mensaje.

Su valor inicial es el valor especial MQPRI_PRIORITY_AS_Q_DEF

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *priority & = MQMessage*.**Prioridad**

Para establecer: *MQMessage*.**Priority** = *prioridad &*

Propiedad PutApplicationName

Lectura-grabación. El PutApplName de MQMD. Forma parte del contexto del origen de mensaje.

Su valor inicial contiene únicamente blancos.

Definido en: clase MQMessage

Tipo de datos: String de 28 caracteres

Sintaxis: Para obtener: *putapplnm $= MQMessage*.**PutApplicationNombre**

Para establecer: *MQMessage*.**PutApplicationName** = *putapplnm $*

Propiedad PutApplicationType

Lectura-grabación. El PutApplType de MQMD. Es parte del contexto del origen de mensaje.

Su valor inicial es MQAT_NO_CONTEXT

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- Consulte PutApplType (MQLONG).

Sintaxis: Para obtener: *putappltp & = MQMessage*.**PutApplicationType**

Para establecer: *MQMessage*.**PutApplicationType** = *putappltp &*

Propiedad PutDateTime

Lectura/Escritura. Esta propiedad combina los campos PutDate y PutTime de MQMD. Se trata de partes del contexto de origen de mensaje que indican cuándo se ha colocado el mensaje.

La extensión ActiveX convierte entre el formato de fecha/hora ActiveX y los formatos de fecha y hora utilizados en un MQMD de WebSphere MQ . Si se recibe un mensaje que tiene una PutDate o PutTime no válida, entonces la propiedad PutDateTime después del método de obtención se establece a EMPTY.

Su valor inicial es EMPTY.

Definido en: clase MQMessage

Tipo de datos: Variant de tipo 7 (date/time) o EMPTY.

Sintaxis: Para obtener: *datetime = MQMessage*.**PutDateTime**

Para establecer: *MQMessage*.**PutDateTime** = *datetime*

Propiedad ReplyToQueueManagerName

Lectura-grabación. El campo ReplyToQMgr del MQMD.

El valor inicial es todo blancos.

Definido en: clase MQMessage

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *replytoqmgr $= MQMessage*.**ReplyToQueueManagerName**

Para establecer: *MQMessage*.**ReplyToQueueManagerName** = *replytoqmgr $*

Propiedad ReplyToQueueName

Lectura-grabación. Campo MsgType del MQMD.

El valor inicial es todo blancos.

Definido en: clase MQMessage

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *replytoq $= MQMessage*.**ReplyToQueueName**

Para establecer: *MQMessage*.**ReplyToQueueName** = *replytoq $*

Propiedad Report

Lectura-grabación. Opciones de informe del mensaje.

Su valor inicial es MQRO_NONE.

Definido en: clase MQMessage

Tipo de datos: Long

Valores:

- Consulte Report.

Sintaxis: Para obtener: *report & = MQMessage*.**Informe**

Para establecer: *MQMessage*.**Informe** = *informe &*

Propiedad TotalMessageLength

Solo lectura. Recupera la longitud del último mensaje recibido por MQGET. Si el mensaje no se ha truncado, este valor es igual al valor de la propiedad MessageLength.

Definido en: clase MQMessage

Tipo de datos: Long

Sintaxis: Para obtener: *totalmessagelength & = MQMessage*.**TotalMessageLength**

Propiedad UserId

Lectura-grabación. El UserIdentifier del MQMD. Forma parte del contexto de identidad del mensaje.

Su valor inicial contiene únicamente blancos.

Definido en: clase MQMessage

Tipo de datos: String de 12 caracteres

Sintaxis: Para obtener: *userid $= MQMessage*.**UserId**

Para establecer: *MQMessage*.**UserId** = *userid $*

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQMessage y MQSession.

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**ClearErrorCodes ()**

Método ClearMessage

Este método borra la parte de búfer de datos del objeto MQMessage. Se perderán todos los datos de mensaje del búfer de datos, porque MessageLength, DataLength y DataOffset se establecen a cero.

La parte del descriptor de mensaje (MQMD) no se ve afectada: una aplicación podría tener que modificar algunos de los campos del MQMD antes de reutilizar el objeto MQMessage. Para establecer los campos MQMD a su valor original, utilice New para sustituir el objeto por una instancia nueva.

Definido en: clase MQMessage

Sintaxis: Llame a *MQMessage*.**ClearMessage()**

Método Read

Lee una secuencia de bytes del búfer del mensaje a un vector de bytes. DataOffset se incrementa y DataLength se decrementa en el número de bytes leídos.

Definida en:
Clase MQMessage

Sintaxis: Data = MQMessage.**Read**(len &)

Parámetros:
len &: Long. Longitud de los datos que se van a leer, en bytes.

Método ReadBoolean

Lee un valor booleano de 1 byte de la posición actual en el búfer de mensajes y devuelve un valor booleano de 2 bytes TRUE(-1)/FALSE(0). DataOffset se incrementa en uno y DataLength se decrementa en uno.

Definida en:
Clase MQMessage

Sintaxis: *value = MQMessage*.**ReadBoolean**

Método ReadByte

Este método lee 1 byte del almacenamiento intermedio de datos de mensaje, empezando por el carácter al que hace referencia DataOffset y lo devuelve como un valor entero entero (2 bytes con signo) en el rango de -128 a 127.

El método falla si MQMessage.DataLength es menor que 1 cuando se emite.

DataOffset se incrementa en 1 y DataLength se decrementa en 1 si el método ejecuta satisfactoriamente.

Se asume que el único byte de datos de mensaje es un entero binario con signo.

Definido en: clase MQMessage

Sintaxis: *integerv% = MQMessage*.**ReadByte**

Método ReadDecimal2

Lee un número decimal empaquetado de 2 bytes y lo devuelve como valor entero de 2 bytes con signo. DataOffset se incrementa en dos y DataLength se decrementa en dos.

Definida en:
Clase MQMessage

Sintaxis: *value% = MQMessage*.**ReadDecimal2**

Método ReadDecimal4

Lee un número decimal empaquetado de 4 bytes y lo devuelve como valor entero de 4 bytes con signo. DataOffset se incrementa en cuatro y DataLength se decrementa en cuatro.

Definida en:
Clase MQMessage

Sintaxis: Llamar *valor & = MQMessage*.**ReadDecimal4**

Método ReadDouble

Este método lee 8 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor de coma flotante Doble (8 bytes con signo).

El método falla si MQMessage.DataLength es menor que 8 cuando se emite.

DataOffset se incrementa en 8 y DataLength se decrementa en 8 si el método ejecuta satisfactoriamente.

Se asume que los 8 caracteres de los datos de mensaje son un número en coma flotante binario.
La codificación se especifica mediante la propiedad MQMessage.Encoding. Tenga en cuenta que la conversión del formato System/360 no está soportada.

Definido en: clase MQMessage

Sintaxis: *doublev# = MQMessage*.**ReadDouble**

Método ReadDouble4

Los métodos ReadDouble4 y WriteDouble4 son alternativas a ReadFloat y WriteFloat. Esto se debe a que dan soporte a valores de mensaje de coma flotante System/390 de 4 bytes que son demasiado grandes para convertirlos al formato de coma flotante IEEE de 4 bytes.

Este método lee 4 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor de coma flotante Doble (8 bytes con signo).

El método falla si MQMessage.DataLength es menor que 4 cuando se emite.

DataOffset se incrementa en cuatro y DataLength se decrementa en cuatro si el método ejecuta satisfactoriamente.

Se asume que los 4 caracteres de los datos de mensaje son un número en coma flotante binario. La codificación se especifica mediante la propiedad MQMessage.Encoding. Tenga en cuenta que la conversión del formato System/360 no está soportada.

Definido en: clase MQMessage

Sintaxis: *doublev# = MQMessage.***ReadDouble4**

Método ReadFloat

Este método lee 4 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor de coma flotante único (con signo de 4 bytes).

El método falla si MQMessage.DataLength es menor que 4 cuando se emite.

DataOffset se incrementa en cuatro y DataLength se decrementa en cuatro si el método ejecuta satisfactoriamente.

Se asume que los 4 caracteres de los datos de mensaje son un número en coma flotante. La codificación se especifica mediante la propiedad MQMessage.Encoding. Tenga en cuenta que la conversión del formato System/360 no está soportada.

Definido en: clase MQMessage

Sintaxis: *singlev! = MQMessage***.ReadFloat**

Método ReadInt2

Este método es idéntico al método ReadShort.

Sintaxis: *integerv% = MQMessage***.ReadInt2**

Método ReadInt4

Este método es idéntico al método ReadLong.

Sintaxis: *bigint & = MQMessage***.ReadInt4**

Método ReadLong

Este método lee 4 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor entero largo (con signo de 4 bytes).

El método falla si MQMessage.DataLength es menor que 4 cuando se emite.

DataOffset se incrementa en cuatro y DataLength se decrementa en cuatro si el método ejecuta satisfactoriamente.

Se asume que los 4 caracteres de los datos de mensaje son entero binario.La codificación se especifica mediante la propiedad MQMessage.Encoding.

Definido en: clase MQMessage

Sintaxis: *bigint & = MQMessage***.ReadLong**

Método ReadNullTerminatedString

Este método se utiliza en lugar de ReadString si la cadena puede incluir caracteres nulos.

Este método lee el número especificado de bytes del búfer de datos del mensaje empezando por el byte referenciado por DataOffset y lo devuelve como una cadena de ActiveX. Si la cadena incorpora un nulo antes del final, la longitud de la cadena devuelta se reduce a fin de reflejar solo los caracteres anteriores al nulo.

DataOffset se incrementa y DataLength se decrementa en el valor especificado, independientemente de si la cadena incluye caracteres nulos.

Se supone que los caracteres de los datos de mensaje son una cadena en la página de códigos especificada en la propiedad MQMessage.CharacterSet. Se hace la conversión a una representación ActiveX para la aplicación.

Definida en:
Clase MQMessage

Sintaxis: *string $ = MQMessage*.**ReadNullTerminatedString(length &)**

Parámetros:
longitud & *Long.* Longitud del campo cadena en bytes

Método ReadShort

Este método lee 2 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor entero (2 bytes con signo).

El método falla si MQMessage.DataLength es menor que 2 cuando se emite.

DataOffset se incrementa en 2 y DataLength se decrementa en 2 si el método ejecuta satisfactoriamente.

Se asume que los 2 caracteres de los datos de mensaje son entero binario. La codificación se especifica mediante la propiedad MQMessage.Encoding.

Definido en: clase MQMessage

Sintaxis: *integerv% = MQMessage***.ReadShort**

método ReadString

Este método lee un número n de bytes del almacenamiento intermedio de datos del mensaje, comenzando por el byte al que hace referencia DataOffset y lo devuelve como una serie de caracteres ActiveX.

El método no se ejecuta correctamente si MQMessage.DataLength es menor que n cuando se emite.

DataOffset se incrementa en n y DataLength disminuye en n si el método se ejecuta correctamente.

Se presupone que los n caracteres de los datos del mensaje son una serie de la página de códigos especificada mediante la propiedad MQMessage.CharacterSet. Se hace la conversión a una representación ActiveX para la aplicación.

Definido en: clase MQMessage

Sintaxis: *stringv $= MQMessage***.ReadString***(length &*)

Parámetro
length & Largo. Longitud del campo cadena en bytes

Método ReadUInt2

Este método lee 2 bytes del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor entero largo (con signo de 4 bytes).

El método falla si MQMessage.DataLength es menor que 2 cuando se emite.

DataOffset se incrementa en 2 y DataLength se decrementa en 2 si el método ejecuta satisfactoriamente.

Se asume que los 2 bytes de los datos de mensaje son un entero binario sin signo. La codificación se especifica mediante la propiedad MQMessage.Encoding.

Definido en: clase MQMessage

Sintaxis: *bigint & = MQMessage***.ReadUInt2**

Método ReadUnsignedByte

Este método lee 1 byte del almacenamiento intermedio de datos de mensaje, empezando por el byte al que hace referencia DataOffset y lo devuelve como un valor entero entero (con signo de 2 bytes) en el rango de 0 a 255.

El método falla si MQMessage.DataLength es menor que 1 cuando se emite.

DataOffset se incrementa en 1 y DataLength se decrementa en 1 si el método ejecuta satisfactoriamente.

Se asume que el único carácter de datos de mensaje es un entero binario sin signo.

Definido en: clase MQMessage

Sintaxis: *integerv% = MQMessage*.**ReadUnsignedByte**

Método ReadUTF

Este método lee una serie de formato UTF del mensaje que empieza por el byte al que hace referencia DataOffset y la devuelve como una serie ActiveX . La serie del mensaje consta de una longitud de 2 bytes seguida de los datos de tipo carácter.

El método falla si MQMessage.DataLength es menor que la longitud de serie cuando se emite.

DataOffset se incrementa en la longitud de serie y DataLength se decrementa en la longitud de serie si el método se ejecuta correctamente.

Definida en:
Clase MQMessage

Sintaxis: *value $= MQMessage*.**ReadUTF**

Método ResizeBuffer

Este método modifica la cantidad de almacenamiento asignada internamente en un momento dado para dar cabida al búfer de datos del mensaje. Proporciona a la aplicación cierto control sobre la gestión automática de búfers. Dicho control consiste en que, si la aplicación sabe que va a tener que manipular un mensaje grande, puede asegurarse de que se le asigna un búfer lo suficientemente grande. La aplicación no necesita utilizar esta llamada; si no lo hace, el código automático de gestión de búfers incrementará tamaño del búfer para que encaje.

Si se cambia el tamaño del búfer para que sea menor que el indicado en MessageLength, se pueden perder datos. Si se pierden datos, el método devuelve el CompletionCode MQCC_WARNING y el ReasonCode MQRC_DATA_TRUNCATED.

Si se cambia el tamaño del búfer para que sea mayor que el valor de la propiedad **DataOffset**, la:

- Propiedad **DataOffset** se modifica para que apunte al final del nuevo búfer.
- La propiedad **DataLength** se establece a cero.
- La propiedad **MessageLength** se establece al nuevo tamaño del búfer.

Definida en:
Clase MQMessage

Sintaxis: *MQMessage*.**ResizeBuffer**(*Length &*)

Parámetro:
Length& Long. Tamaño necesario en caracteres.

Método Write

Escribe una secuencia de bytes en el búfer de mensaje procedentes de un vector de bytes en la posición referenciada por DataOffset. Si es necesario, la longitud del búfer (MQMessage.MQMessageLength) se amplía para dar cabida a la longitud completa del vector de bytes. DataOffset se incrementa en el número de bytes escritos si el método ejecuta satisfactoriamente.

Definida en:
Clase MQMessage

Sintaxis: Llamar a *MQMessage*.**Write**(valor)

Parámetros:
value: vector de bytes o referencia variante a un vector de bytes

Método WriteBoolean

Escribe un valor booleano de 1 byte en la posición actual del búfer de mensaje a partir de un valor booleano de 2 bytes. DataOffset se incrementa en uno.

Definida en:
Clase MQMessage

Sintaxis: Llame a *MQMessage*.**WriteBoolean**(valor)

Parámetro:
value: booleano (2 bytes). Valor que hay que escribir.

Método WriteByte

Este método recibe un valor entero de 2 bytes con signo y lo escribe en el búfer de datos de mensaje como un número binario de 1 byte referenciado por DataOffset. Sustituye los datos que ya haya en esa posición del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en 1 si el método ejecuta correctamente.

El valor especificado tiene que pertenecer al rango -128 a 127. En caso contrario, el método devolverá el código de terminación (CompletionCode) MQCC_FAILED y el código de razón (ReasonCode) MQRC_WRITE_VALUE_ERROR.

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteByte**(*valor%*)

Parámetro: *value%* Integer. Valor que hay que escribir.

Método WriteDecimal2

Escribe un entero de 2 bytes con signo como un número decimal empaquetado de 2 bytes. DataOffset se incrementa en 2.

Definida en:
Clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteDecimal2**(valor%)

Parámetro:
value% Integer. Valor que hay que escribir.

Método WriteDecimal4

Escribe un entero de 4 bytes con signo como un número decimal empaquetado de 4 bytes. DataOffset se incrementa en cuatro.

Definida en:
Clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WritedDecimal4**(valor &)

Parámetro:
valor & Largo. Valor que hay que escribir.

Método WriteDouble

Este método recibe un valor en coma flotante de 8 bytes con signo y lo escribe en el búfer de datos de mensaje como un número en coma flotante de 8 bytes que empieza en la posición referenciada por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en 8 si el método ejecuta correctamente.

Este método efectúa una conversión a la representación en coma flotante especificada por la propiedad MQMessage.Encoding. *La conversión al formato System/360 no está soportada.*

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteDouble**(*value#*)

Parámetro:
value# Double. Valor que hay que escribir.

Método WriteDouble4

Consulte "Método ReadDouble4" en la página 1103 para obtener una descripción de cuándo hay que utilizar ReadDouble4 y WriteDouble4 en lugar de ReadFloat y WriteFloat.

Este método recibe un valor en coma flotante de 8 bytes con signo y lo escribe en el búfer de datos de mensaje como un número en coma flotante de 4 bytes que empieza en la posición referenciada por DataOffset.

DataOffset se incrementa en cuatro si el método ejecuta correctamente.

Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

Este método efectúa una conversión a la representación en coma flotante especificada por la propiedad MQMessage.Encoding. *La conversión al formato System/360 no está soportada.*

Definido en: clase MQMessage

Sintaxis: Llame a *MQMessage*.**WriteDouble4 (value#)**

Parámetro: *value#* Double. Valor que hay que escribir.

Método WriteFloat

Este método recibe un valor en coma flotante de 4 bytes con signo y lo escribe en el búfer de datos de mensaje como un número en coma flotante de 4 bytes que empieza en el carácter referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en cuatro si el método ejecuta correctamente.

Este método efectúa una conversión a la representación binaria especificada por la propiedad MQMessage.Encoding. *La conversión al formato System/360 no está soportada.*

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteFloat**(*valor!*)

Parámetro *value!* Float. Valor que hay que escribir.

Método WriteInt2

Este método es idéntico al método WriteShort.

Sintaxis: Llamar a *MQMessage*.**WriteInt2**(*valor%*)

Parámetro *value%* Integer. Valor que hay que escribir.

Método WriteInt4

Este método es idéntico al método WriteLong.

Sintaxis: Llamar a *MQMessage*.**WriteInt4**(*valor &*)

Parámetro *value &* Long. Valor que hay que escribir.

Método WriteLong

Este método recibe un valor entero de 4 bytes con signo y lo escribe en el búfer de datos de mensaje como un número binario de 4 bytes que empieza en el byte referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en cuatro si el método ejecuta correctamente.

Este método efectúa una conversión a la representación binaria especificada por la propiedad MQMessage.Encoding.

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteLong**(*value &*)

Parámetro *value &* Long. Valor que hay que escribir.

Método WriteNullTerminatedString

Este método lleva a cabo un WriteString normal y rellena cualquier byte restante hasta la longitud especificada con nulos. Si el número de bytes escritos por la cadena de escritura inicial es igual a la longitud especificada, no se escribirá ningún nulo. Si el número de bytes sobrepasa la longitud especificada, se establecerá un error (código de razón MQRC_WRITE_VALUE_ERROR).

DataOffset se incrementa en la longitud especificada si el método se ejecuta correctamente.

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteNullTerminatedString***(value$, length &)*

Parámetros:
value$ String. Valor que hay que escribir.

length& Long. Longitud del campo cadena en bytes

Método WriteShort

Este método recibe un valor entero de 2 bytes con signo y lo escribe en el búfer de datos de mensaje como un número binario de 2 bytes que empieza en el byte referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en 2 si el método ejecuta correctamente.

Este método efectúa una conversión a la representación binaria especificada por la propiedad MQMessage.Encoding.

Definido en: clase MQMessage

Sintaxis: Llame a *MQMessage*.**WriteShort**(*valor%*)

Parámetro *value%* Integer. Valor que hay que escribir.

Método WriteString

Este método recibe una cadena ActiveX y la escribe en el búfer de datos de mensaje empezando en el byte referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa por la longitud de la cadena en bytes si el método ejecuta correctamente.

El método convierte los caracteres a la página de códigos especificada por la propiedad MQMessage.CharacterSet.

Definido en: clase MQMessage

Sintaxis: Llame a *MQMessage*.**WriteString**(*value $*)

Parámetro *value$* String. Valor que hay que escribir.

Método WriteUInt2

Este método recibe un valor entero de 4 bytes con signo y lo escribe en el búfer de datos de mensaje como un número binario de 2 bytes sin signo que empieza en el byte referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en 2 si el método ejecuta correctamente.

Este método efectúa una conversión a la representación binaria especificada por la propiedad MQMessage.Encoding. El valor especificado tiene que encontrarse en el rango de 0 a 2**16-1. En caso contrario, el método devolverá el código de terminación (CompletionCode) MQCC_FAILED y el código de razón (ReasonCode) MQRC_WRITE_VALUE_ERROR.

Definido en: clase MQMessage

Sintaxis: Llamar a *MQMessage*.**WriteUInt2**(*valor &*)

Parámetro *value &* Long. Valor que hay que escribir.

Método WriteUnsignedByte

Este método recibe un valor entero de 2 bytes con signo y lo escribe en el búfer de datos de mensaje como un número binario de 1 byte sin signo que empieza en el carácter referenciado por DataOffset. Sustituye los datos que ya haya en esas posiciones del búfer y amplía la longitud del mismo (MQMessage.MessageLength) en caso de ser necesario.

DataOffset se incrementa en 1 si el método ejecuta correctamente.

El valor especificado debe estar comprendido entre 0 y 255. En caso contrario, el método devolverá el código de terminación (CompletionCode) MQCC_FAILED y el código de razón (ReasonCode) MQRC_WRITE_VALUE_ERROR.

Definida en:
Clase MQMessage

Sintaxis: Call *MQMessage*.**WriteUnsignedByte**(*valor%*)

Parámetro *value%* Integer. Valor que hay que escribir.

Método WriteUTF

Este método recibe una cadena de ActiveX y la escribe en el búfer de datos de mensaje en la posición actual en formato UTF. Los datos escritos tienen una longitud de 2 bytes seguidos de los datos de tipo carácter. DataOffset se incrementa por la longitud de la cadena si el método ejecuta correctamente.

Definida en:
Clase MQMessage

Sintaxis: Llamada *MQMessage*.**WriteUTF**(value$)

Parámetro:
value$ String. Valor que hay que escribir.

Clase MQPutMessageOptions

Esta clase encapsula las diversas opciones que controlan la acción de colocar un mensaje en una cola de WebSphere MQ .

Contención

La clase MQPutMessageOptions está contenida en la clase MQSession.

Creación

New crea un nuevo objeto MQPutMessageOptions y establece todas las propiedades en sus valores iniciales.

También puede utilizarse el método AccessPutMessageOptions de la clase MQSession.

Sintaxis

Dim *pmo* **As New MQPutMessageOptions** o bien

Set *pmo* = **New MQPutMessageOptions**

Propiedades

- "Propiedad CompletionCode" en la página 1111.
- "Propiedad Options" en la página 1111.
- "propiedad ReasonCode" en la página 1111.
- "propiedad ReasonName" en la página 1112.
- "Propiedad RecordFields" en la página 1112.
- "Propiedad ResolvedQueueManagerName" en la página 1112.
- "Propiedad ResolvedQueueName" en la página 1112.

Métodos

- "Método ClearErrorCodes" en la página 1112.

Propiedad CompletionCode

Solo lectura. Devuelve el código de terminación establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: Clase MQPutMessageOptions

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = PutOpts***.CompletionCode**

Propiedad Options

Lectura-grabación. El campo Options de MQPMO. El valor inicial de este campo es MQPMO_NONE. Para obtener más información, consulte Opciones de MQPMO.

Definido en: clase MQPutMessageOptions.

Tipo de datos: Long

Sintaxis: Para obtener: *options & = PutOpts***.Opciones**

Para establecer: *PutOpts***.Opciones** = *opciones &*

Las opciones MQPMO_PASS_IDENTITY_CONTEXT y MQPMO_PASS_ALL_CONTEXT no están soportadas.

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: Clase MQPutMessageOptions

Tipo de datos: Long

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = PutOpts*.**ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido en: Clase MQPutMessageOptions

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= PutOpts*.**ReasonName**

Propiedad RecordFields

Lectura-grabación. Distintivos que indican qué campos se van a personalizar en cada cola cuando se coloca un mensaje en una lista de distribución. El valor inicial es cero.

Esta propiedad corresponde a los distintivos PutMsgRecFields de la estructura MQI MQPMO. En la MQI, estos distintivos controlan qué campos (en la estructura MQPMR) están presentes y son usados por la MQPUT. En un objeto MQPutMessageOptions, estos campos siempre están presentes y, por tanto, los distintivos solo afectan a qué campos utiliza la operación de colocación. Consulte la publicación *WebSphere MQ Application Programming Reference* para obtener más detalles.

Definida en:
Clase MQPutMessageOptions

Tipo de datos:
Long

Sintaxis: Para obtener: *recordfields & = PutOpts*.**RecordFields**

Para establecer: *PutOpts*.**RecordFields** = *recordfields &*

Propiedad ResolvedQueueManagerName

Solo lectura. El campo ResolvedQMgrName de MQPMO. Consulte ResolvedQMgrName (MQCHAR48) para obtener los detalles. Su valor inicial contiene únicamente blancos.

Definido en: Clase MQPutMessageOptions

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *qmgr $= PutOpts*.**ResolvedQueueManagerName**

Propiedad ResolvedQueueName

Solo lectura. El campo ResolvedQName de MQPMO. Consulte ResolvedQName (MQCHAR48) para obtener los detalles. Su valor inicial contiene únicamente blancos.

Definido en: Clase MQPutMessageOptions

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *qname $= PutOpts*.**ResolvedQueueNombre**

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQPutMessageOptions y MQSession.

Definido en: Clase MQPutMessageOptions

Sintaxis: Call *PutOpts*.**ClearErrorCodes ()**

clase MQGetMessageOptions

Esta clase encapsula las diversas opciones que controlan la acción de obtener un mensaje de una cola WebSphere MQ .

Contención

La clase MQGetMessageOptions está contenida en la clase MQSession.

Creación

New crea un nuevo objeto MQGetMessageOptions y establece todas las propiedades en sus valores iniciales.

También puede utilizarse el método AccessGetMessageOptions de la clase MQSession.

Propiedades

- "Propiedad CompletionCode" en la página 1113
- "Propiedad MatchOptions" en la página 1114
- "Propiedad Options" en la página 1114
- "propiedad ReasonCode" en la página 1114
- "propiedad ReasonName" en la página 1114
- "Propiedad ResolvedQueueName" en la página 1114
- "Propiedad WaitInterval" en la página 1114

Métodos

- "Método ClearErrorCodes" en la página 1115

Sintaxis

Dim *gmo* **As New MQGetMessageOptions** o bien

Set *gmo* = **New MQGetMessageOptions**

Propiedad CompletionCode

Solo lectura. Devuelve el código de terminación establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: clase MQGetMessageOptions.

Tipo de datos: Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = GetOpts*.**CompletionCode**

Propiedad MatchOptions

Lectura-grabación. Opciones que controlan los criterios de selección utilizados para MQGET. El valor inicial es MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID.

Definida en:
clase MQGetMessageOptions

Tipo de datos:
Long

Valores:
Consulte MatchOptions (MQLONG).

Sintaxis: Para obtener: *matchoptions & = GetOpts*.**MatchOptions**

Para establecer: *GetOpts*.**MatchOptions** =*matchoptions &*

Propiedad Options

Lectura-grabación. El campo Options de MQPMO. Consulte Options para obtener detalles. El valor inicial es MQGMO_NO_WAIT.

Definido en: clase MQGetMessageOptions.

Tipo de datos: Long

Sintaxis: Para obtener: *options & = GetOpts*.**Opciones** Para establecer: *GetOpts*.**Opciones** = *opciones &*

propiedad ReasonCode

Solo lectura. Devuelve el código de razón establecido por el último método o acceso de propiedad emitido contra el objeto.

Definido en: Clase MQGetMessageOptions

Tipo de datos: Long

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = GetOpts*.**ReasonCode**

propiedad ReasonName

Solo lectura. Devuelve el nombre simbólico del código de razón más reciente. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE". **Definido en:** Clase MQGetMessageOptions

Tipo de datos: String

Valores:

- Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQGetMessageOptions*.**ReasonName**

Propiedad ResolvedQueueName

Solo lectura. El campo ResolvedQName de MQPMO. Consulte ResolvedQName (MQCHAR48) para obtener los detalles. Su valor inicial contiene únicamente blancos.

Definido en: Clase MQGetMessageOptions

Tipo de datos: String de 48 caracteres

Sintaxis: Para obtener: *qname $= GetOpts*.**ResolvedQueueNombre**

Propiedad WaitInterval

Lectura/Escritura. El campo WaitInterval de MQGMO. Tiempo máximo, en milisegundos, que una operación de obtención espera a que llegue un mensaje, en caso de que en la propiedad Options se haya solicitado la opción de espera. Este campo tiene un valor inicial de 0. Para obtener detalles sobre las opciones MQGMO, consulte MQGMO.

Definido en: Clase MQGetMessageOptions

Tipo de datos: Long

Sintaxis: Para obtener: *wait & = GetOpts*.**WaitInterval**

Para establecer: *GetOpts*.**WaitInterval** = wait &

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQGetMessageOptions y MQSession.

Definido en: Clase MQGetMessageOptions

Sintaxis: Llamada *GetOpts*.**ClearErrorCodes ()**

Clase MQDistributionList

Esta clase encapsula una colección de colas (locales, remotas o alias) para la salida.

Creación

New crea un objeto MQDistributionList nuevo.

También puede utilizarse el método AddDistributionList de la clase MQQueueManager.

Propiedades

- "Propiedad AlternateUserId" en la página 1115
- "Propiedad CloseOptions" en la página 1116
- "Propiedad CompletionCode" en la página 1116
- "propiedad ConnectionReference" en la página 1116
- "Propiedad FirstDistributionListItem" en la página 1116
- "propiedad IsOpen" en la página 1117
- "Propiedad OpenOptions" en la página 1117
- "propiedad ReasonCode" en la página 1117
- "propiedad ReasonName" en la página 1117

Método

- "Método AddDistributionListItem" en la página 1118
- "Método ClearErrorCodes" en la página 1118
- "Método Close" en la página 1118
- "Método Open" en la página 1118
- "Método Put" en la página 1118

Sintaxis

Dim *distlist.***As New MQDistributionList** o **Set** *distlist* = **New MQDistributionList**

Propiedad AlternateUserId

Lectura-grabación. El ID del usuario alternativo utilizado para validar el acceso a la lista de colas cuando se abren.

Definida en:
Clase MQDistributionList

Tipo de datos:
Serie de 12 caracteres

Sintaxis: Para obtener: *altuser $= MQDistributionList*.**AlternateUserId**

Para establecer: *MQDistributionList*.**AlternateUserId** = *altuser $*

Propiedad CloseOptions

Lectura-grabación. Opciones utilizadas para controlar lo que sucede cuando se cierra la lista de distribución. El valor inicial es MQCO_NONE.

Definida en:
Clase MQDistributionList

Tipo de datos:
Long

Valores:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

Sintaxis: Para obtener: *closeopt & = MQDistributionList*.**CloseOptions**

Para establecer: *MQDistributionList*.**CloseOptions** = *closeopt &*

Propiedad CompletionCode

Solo lectura. El código de terminación definido por el último método o acceso a propiedad emitido para el objeto.

Definida en:
Clase MQDistributionList

Tipo de datos:
Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode & = MQDistributionList*.**CompletionCode**

propiedad ConnectionReference

Lectura-grabación. El gestor de colas al que pertenece la lista de distribución.

Definida en:
Clase MQDistributionList

Tipo de datos:
MQQueueManager

Sintaxis: Para obtener: *set queuemanager = MQDistributionList*.**ConnectionReference**

Para definir: *set MQDistributionList*. **ConnectionReference** = *queuemanager*

Propiedad FirstDistributionListItem

Solo lectura. El primer objeto del elemento de lista de distribución asociado con la lista de distribución.

Definida en:
Clase MQDistributionList

Tipo de datos:
MQDistributionListItem

Valores:

Sintaxis: Para obtener: *set distributionlistitem = MQDistributionList*.**FirstDistributionListItem**

propiedad IsOpen

Solo lectura.

Definida en:
Clase MQDistributionList

Tipo de datos:
Boolean

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxis: Para obtener: *IsOpen = MQDistributionList*.**IsOpen**

Propiedad OpenOptions

Lectura-grabación. Opciones que deben utilizarse cuando se abre la lista de distribución.

Definida en:
Clase MQDistributionList

Tipo de datos:
Long

Valores:
Consulte Opciones de MQPMO.

Sintaxis: Para obtener: *openopt & = MQDistributionList*.**OpenOptions**

Para establecer: *MQDistributionList.***OpenOptions** = openopt &

propiedad ReasonCode

Solo lectura. El código de razón definido por el último acceso a un método o propiedad emitido para el objeto.

Definida en:
Clase MQDistributionList

Tipo de datos:
Long

Valores:
Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasoncode & = MQDistributionList*.**ReasonCode**

propiedad ReasonName

Solo lectura. Es el nombre simbólico de ReasonCode. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definida en:
Clase MQDistributionList

Tipo de datos:
Cadena

Valores:
Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQDistributionList*.**ReasonName**

Método AddDistributionListItem

Crea un nuevo objeto de elemento MQDistributionListy lo asocia con el objeto de lista de distribución. El parámetro de nombre de cola es obligatorio.

La propiedad DistributionList del elemento de lista de distribución se establece en la lista de distribución propietaria y la propiedad FirstDistributionListItem de la lista de distribución se establece para hacer referencia a este nuevo elemento de lista de distribución.

Para el nuevo elemento de lista de distribución, la propiedad PreviousDistributionListItem no se establece en nada y la propiedad NextDistributionListItem se establece para hacer referencia a cualquier elemento de lista de distribución que anteriormente era el primero, o a nada si no había ninguno anteriormente (es decir, el nuevo se inserta delante de los que ya existen).

Esto devolverá un error si la lista de distribución está abierta.

Definida en:
Clase MQDistributionList

Sintaxis: set distributionlistitem = *MQDistributionList*.**AddDistributionListItem** (QName$, QMgrName$)

Parámetros:

QName$ String. Nombre de la cola de WebSphere MQ .

QMgrName$ String. Nombre del gestor de colas de WebSphere MQ .

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQDistributionList y MQSession.

Definida en:
Clase MQDistributionList

Sintaxis: Llamar a *MQDistributionList*.**ClearErrorCodes**()

Método Close

Cierra una lista de distribución utilizando el valor actual de las opciones de Close.

Definida en:
Clase MQDistributionList

Sintaxis: Llamar a *MQDistributionList*.**Cerrar**()

Método Open

Abre cada una de las colas especificadas por el QueueName y (si procede) QueueManagerlas propiedades de nombre de los elementos de lista de distribución asociados con el objeto actual utilizando el valor actual de AlternateUserId.

Definida en:
Clase MQDistributionList

Sintaxis: Llamar a *MQDistributionList*.**Abrir**()

Método Put

Coloca un mensaje en cada una de las colas identificadas por los elementos de la lista de distribución asociados con la lista de distribución.

Definida en:

Clase MQDistributionList

Sintaxis

Llame a MQDistributionList.**Put**(Message, PutMsgOptions &)

Parámetros

Message: objeto MQMessage que representa al mensaje que se va a colocar.

PutMsgOptions: objeto MQPutMessageOptions que contiene opciones para controlar la operación de transferencia. Si no se especifica, se utilizan las PutMessageOptions predeterminadas.

Este método utiliza un objeto MQMessage como parámetro. Las siguientes propiedades de elemento de lista de distribución se pueden alterar como resultado de este método:

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Comentarios
- AccountingToken
- AccountingTokenHex

Clase MQDistributionListItem

Esta clase encapsula las estructuras MQOR, MQRR y MQPMR y las asocia con una lista de distribución de propietario.

Creación

Utilice el método AddDistributionListItem de la clase MQDistributionList.

Propiedades

Métodos

- “propiedad AccountingToken” en la página 1120.
- “Propiedad AccountingTokenHex” en la página 1121.
- “Propiedad CompletionCode” en la página 1121.
- “Propiedad CorrelationId” en la página 1121.
- “Propiedad CorrelationIdHex” en la página 1121.
- “propiedad DistributionList” en la página 1122.
- “propiedad Feedback” en la página 1122.
- “propiedad GroupId” en la página 1122.
- “Propiedad GroupIdHex” en la página 1122.

- “propiedad MessageId” en la página 1123.
- “Propiedad MessageIdHex” en la página 1123.
- “propiedad NextDistributionListItem” en la página 1123.
- “propiedad PreviousDistributionListItem” en la página 1123.
- “Propiedad QueueManagerName” en la página 1124.
- “propiedad QueueName” en la página 1124.
- “propiedad ReasonCode” en la página 1124.
- “propiedad ReasonName” en la página 1124.
- “Método ClearErrorCodes” en la página 1124.

Propiedades:

- propiedad AccountingToken
- Propiedad AccountingTokenHex
- Propiedad CompletionCode
- Propiedad CorrelationId
- Propiedad CorrelationIdHex
- propiedad DistributionList
- propiedad Feedback
- propiedad GroupId
- Propiedad GroupIdHex
- propiedad MessageId
- Propiedad MessageIdHex
- propiedad NextDistributionListItem
- propiedad PreviousDistributionListItem
- Propiedad QueueManagerName
- propiedad QueueName
- propiedad ReasonCode
- propiedad ReasonName

Métodos:

- Método ClearErrorCodes

Creation:

Utilice el método AddDistributionListItem de la clase MQDistributionList.

propiedad AccountingToken

Lectura-grabación. La propiedad AccountingToken que debe incluirse en la MQPMR de un mensaje cuando se transfiere a la cola. Su valor inicial es todo nulos.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 32 caracteres

Sintaxis: Para obtener: *accountingtoken $= MQDistributionListElemento*.**AccountingToken**

Para establecer: *MQDistributionListElemento*.**AccountingToken** = *accountingtoken $*

Propiedad AccountingTokenHex

Lectura-grabación. La propiedad AccountingToken que debe incluirse en la MQPMR de un mensaje cuando se transfiere a la cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 64 caracteres hexadecimales válidos.

Su valor inicial es "0...0".

Definida en:
Clase MQDistributionListItem

Tipo de datos:
String de 64 caracteres hexadecimales que representan 32 caracteres ASCII.

Sintaxis: Para obtener: *accountingtokenh $= MQDistributionListElemento*.**AccountingTokenHex**

Para establecer: *MQDistributionListElemento*.**AccountingTokenHex** = *accountingtokenh $*

Propiedad CompletionCode

Solo lectura. El código de terminación definido por la última petición de apertura o transferencia emitida para el objeto de la lista de distribución propietaria.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *completioncode $= MQDistributionListElemento*.**CompletionCode**

Propiedad CorrelationId

Lectura-grabación. El CorrelId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola. Su valor inicial es todo nulos.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 24 caracteres

Sintaxis: Para obtener: *correlid $= MQDistributionListElemento*.**CorrelationId**

Para establecer: *MQDistributionListElemento*.**CorrelationId** = *correlid $*

Propiedad CorrelationIdHex

Lectura-grabación. El CorrelId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0..0".

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Cadena de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *correlidh $= MQDistributionListElemento*.**CorrelationIdHex**

Para establecer: *MQDistributionListElemento*.**CorrelationIdHex** = *correlidh $*

propiedad DistributionList

Solo lectura. La lista de distribución con la que está asociado este elemento de la lista de distribución.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
MQDistributionList

Sintaxis: Para obtener: *set distributionlist = MQDistributionListElemento*.**DistributionList**

propiedad Feedback

Lectura-grabación. El valor de Feedback que debe incluirse en la MQPMR de un mensaje cuando se transfiere a una cola.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Long

Valores:
Consulte Feedback (MQLONG).

Sintaxis: Para obtener: *feedback & = MQDistributionListItem*.**Feedback**

Para establecer: *MQDistributionListItem*.**Feedback** = *feedback &*

propiedad GroupId

Lectura-grabación. El GroupId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola. Su valor inicial es todo nulos.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 24 caracteres

Sintaxis: Para obtener: *groupid $= MQDistributionListElemento*.**GroupId**

Para establecer: *MQDistributionListElemento*.**GroupId** = *groupid $*

Propiedad GroupIdHex

Lectura-grabación. El GroupId que hay que incluir en el MQPMR de un mensaje cuando se coloca en una cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0..0".

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Cadena de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *groupidh $= MQDistributionListElemento*.**GroupIdHex**

Para establecer: *MQDistributionListElemento*.**GroupIdHex** = *groupidh $*

propiedad MessageId

Lectura-grabación. La propiedad MessageId que debe incluirse en la MQPMR de un mensaje cuando se transfiere a la cola. Su valor inicial es todo nulos.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 24 caracteres

Sintaxis: Para obtener: *messageid $= MQDistributionListElemento*.**MessageId**

Para establecer: *MQDistributionListElemento*.**MessageId** = *messageid $*

Propiedad MessageIdHex

Lectura-grabación. La propiedad MessageId que debe incluirse en la MQPMR de un mensaje cuando se transfiere a la cola.

Cada dos caracteres de la cadena representan el equivalente hexadecimal de un solo carácter ASCII. Por ejemplo, el par de caracteres "6" y "1" representan un solo carácter "A", el par de caracteres "6" y "2" representan un solo carácter "B", y así sucesivamente.

Hay que proporcionar 48 caracteres hexadecimales válidos.

Su valor inicial es "0..0".

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Cadena de 48 caracteres hexadecimales que representan 24 caracteres ASCII.

Sintaxis: Para obtener: *messageidh $= MQDistributionListElemento*.**MessageIdHex**

Para establecer: *MQDistributionListElemento*.**MessageIdHex** = *messageidh $*

propiedad NextDistributionListItem

Solo lectura. El siguiente objeto de elemento de lista de distribución asociado con la misma lista de distribución.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
MQDistributionListItem

Sintaxis: Para obtener: *set distributionlistitem = MQDistributionListElemento.***NextDistributionListItem**

propiedad PreviousDistributionListItem

Solo lectura. El objeto anterior del elemento de lista de distribución asociado con la misma lista de distribución.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
MQDistributionListItem

Sintaxis: Para obtener: *set distributionlistitem = MQDistributionListItem*.**PreviousDistributionListItem**

Propiedad QueueManagerName

Lectura-grabación. El nombre del gestor de colas de WebSphere MQ .

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 48 caracteres.

Sintaxis: Para obtener: *qmname $= MQDistributionListItem*.**QueueManagerName**

Para establecer: *MQDistributionListElemento*.**QueueManagerName** = *qmname $*

propiedad QueueName

Lectura-grabación. El nombre de cola de WebSphere MQ .

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Serie de 48 caracteres.

Sintaxis: Para obtener: *qname $= MQDistributionListElemento*.**QueueName**

Para establecer: *MQDistributionListElemento*.**QueueName** = *qname $*

propiedad ReasonCode

Solo lectura. El código de terminación establecido por la última apertura o colocación emitida al objeto de lista de distribución propietario.

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Long

Valores:
Consulte Códigos de razón de API.

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxis: Para obtener: *reasoncode & = MQDistributionListElemento*.**ReasonCode**

propiedad ReasonName

Solo lectura. Es el nombre simbólico de ReasonCode. Por ejemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definida en:
Clase MQDistributionListItem

Tipo de datos:
Cadena

Valores:
Consulte Códigos de razón de API.

Sintaxis: Para obtener: *reasonname $= MQDistributionListElemento*.**ReasonName**

Método ClearErrorCodes

Restablece el CompletionCode a MQCC_OK y el ReasonCode a MQRC_NONE en la clases MQDistributionListItem y MQSession.

Definida en:
Clase MQDistributionListItem

Sintaxis: Llamar a *MQDistributionListElemento*.**ClearErrorCódigos**

Resolución de problemas

Información sobre el recurso de rastreo proporcionado, problemas comunes y ayuda sobre cómo evitarlos.

En la siguiente sección se explica el recurso de rastreo proporcionado y se detallan los problemas comunes, con ayuda para evitarlos:

- "Utilización del recurso de rastreo" en la página 1125
- "Cuando falla el script de WebSphere MQ Automation Classes for ActiveX" en la página 1126
- "códigos de razón" en la página 1126
- "Herramienta de nivel de código" en la página 1129

Utilización del recurso de rastreo

MQAX incluye un recurso de rastreo para ayudar a la organización de servicio a identificar lo que sucede cuando tiene un problema. Muestra las vías de acceso tomadas al ejecutar el script MQAX. A menos que tenga un problema, ejecute con el rastreo desactivado para evitar cualquier uso innecesario de los recursos del sistema.

Para controlar el rastreo se definen tres variables de entorno:

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

Tenga en cuenta que al especificar *any* valor para OMQ_TRACE se activa la facildad de rastreo. Incluso si establece OMQ_TRACE en OFF, el rastreo sigue activo.

Para desactivar el rastreo, no especifique un valor para OMQ_TRACE.

1. Pulse **Iniciar**
2. Pulse **Panel de control**
3. Efectúe una doble pulsación en **Sistema**
4. Pulse **Avanzado**
5. Pulse **Entorno**
6. En la sección titulada "Variables de usuario para (nombre de usuario)", pulse **Nuevo**
7. Entre el nombre de variable y un valor válido en los campos apropiados y pulse **Aceptar**
8. Pulse **Aceptar** para cerrar la ventana Variables de entorno
9. Pulse **Aceptar** para cerrar la ventana Propiedades del sistema
10. Cierre la ventana Panel de control

Al decidir dónde desea que se graben los archivos de rastreo, asegúrese de que tiene autorización suficiente para grabar en el disco, no sólo para leer desde él.

Con el rastreo activado, ralentiza la ejecución de MQAX, pero no afecta al rendimiento de los entornos ActiveX o WebSphere MQ . Cuando ya no necesite un archivo de rastreo, podrá suprimirlo.

Debe detener la ejecución de MQAX para cambiar el estado de la variable OMQ_TRACE.

Nombre y directorio del archivo de rastreo

El nombre del archivo de rastreo toma el formato OMQnnnnn.trc, donde nnnnn es el ID del proceso ActiveX que se ejecuta en ese momento.

Tabla 157. Mandatos y sus efectos

Mandato	Efecto
SET OMQ_TRACE_PATH=unidad:\directorio	Establece el directorio de rastreo donde se escribirá el archivo de rastreo.
SET OMQ_TRACE_PATH =	Elimina la variable de entorno OMQ_PATH en la que se utiliza el directorio de trabajo actual (cuando se inicia ActiveX).
ECHO %OMQ_TRACE_PATH%	Muestra el valor actual del directorio de rastreo en Windows.
SET OMQ_TRACE = xxxxxxxx	Esto establece el rastreo en ON. El rastreo se activa colocando uno o más caracteres después del signo '='. Por ejemplo: SET OMQ_TRACE=yes SET OMQ_TRACE = no. En ambos ejemplos, el rastreo se establecerá en ON. Esto sólo es efectivo para una única ventana/sesión
SET OMQ_TRACE=	Establece el rastreo en OFF
ECHO %OMQ_TRACE%	Muestra el contenido de la variable de entorno en Windows.
set	Muestra el contenido de todas las variables de entorno en Windows.
SET OMQ_TRACE_LEVEL = 9	Establece el nivel de rastreo en 9. Los valores mayores que 9 no generan ninguna información adicional en el archivo de rastreo.

Cuando falla el script de WebSphere MQ Automation Classes for ActiveX

Si el script de WebSphere MQ Automation Classes for ActiveX falla, hay varias fuentes de información.

Informe de síntoma de primer fallo

Al margen del recurso de rastreo, en el caso de errores internos de sistema, podría generarse un informe de síntoma de primer fallo.

Este informe se encuentra en un archivo denominado OMQnnnnn.fdc, donde nnnnn es el número del proceso ActiveX que se ejecuta en ese momento. Encontrará este archivo en el directorio de trabajo desde el que haya iniciado ActiveX o en la ruta especificada en la variable de entorno OMQ_PATH.

Otras fuentes de información

WebSphere MQ proporciona diversos registros de errores e información de rastreo, en función de la plataforma implicada. Consulte el registro de sucesos de aplicación de Windows NT.

códigos de razón

Se pueden producir los siguientes códigos de razón además de los documentados para la MQI de WebSphere MQ . Para otros códigos, consulte el registro de sucesos de la aplicación WebSphere MQ .

Tabla 158. Códigos de razón y lo que significan

Código de razón	Explicación
MQRC_LIBRARY_LOAD_ERROR (6000)	No se han podido cargar una o más de las bibliotecas de WebSphere MQ . Compruebe que todas las bibliotecas de WebSphere MQ estén en la vía de acceso de búsqueda correcta en el sistema que está utilizando. Por ejemplo, asegúrese de que los directorios que contienen las bibliotecas de WebSphere MQ están en PATH.
MQRC_CLASS_LIBRARY_ERROR (6001)	Una de las llamadas a la biblioteca de clases WebSphere MQ ha devuelto un ReasonCode/CompletionCodeinesperado. Compruebe si hay más detalles en el informe de síntomas de primera anomalía. Tome nota del último método/propiedad y clase que se está utilizando e informe al soporte de IBM del problema.
MQRC_STRING_LENGTH_TOO_BIG (6002)	Se ha intentado escribir en el búfer una cadena en formato UTF de longitud superior a 65.535 bytes.

Tabla 158. Códigos de razón y lo que significan (continuación)

Código de razón	Explicación
MQRC_WRITE_VALUE_ERROR (6003)	Se utiliza un valor que está fuera de rango; por ejemplo, msg.WriteByte (240).
MQRC_PACKED_DECIMAL_ERROR (6004)	Se ha intentado leer un número decimal empaquetado del almacenamiento intermedio de mensajes, pero los datos del puntero de datos no están en un formato de datos empaquetado válido.
MQRC_FLOAT_CONVERSION_ERROR (6005)	Se ha intentado leer un número en coma flotante simple o doble del búfer de mensaje, pero los datos del puntero de datos no tienen un formato adecuado de coma flotante.
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	Un objeto abierto no tiene las **OpenOptions** correctas y requiere una o más opciones adicionales. Es necesario volver a abrir el objeto de forma implícita pero el cierre se ha impedido. Establezca **OpenOptions**explícitamente para cubrir todas las eventualidades de modo que no sea necesaria la reapertura implícita. Se ha impedido el cierre porque la cola está abierta para entrada exclusiva y el cierre presentaría una ventana de oportunidad para que otros potencialmente obtuvieran acceso a la cola.
MQRC_REOPEN_INQUIRE_ERROR (6101)	Un objeto abierto no tiene las OpenOptions correctas y requiere una o más opciones adicionales. Es necesario volver a abrir el objeto de forma implícita pero el cierre se ha impedido. Establezca OpenOptions explícitamente para incluir MQOO_INQUIRE. Se ha impedido el cierre porque es necesario comprobar dinámicamente una o más características del objeto antes del cierre, y las OpenOptions todavía no incluyen MQOO_INQUIRE.
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Un objeto abierto no tiene las OpenOptions correctas y requiere una o más opciones adicionales. Es necesario volver a abrir el objeto de forma implícita pero el cierre se ha impedido. Establezca las OpenOptions explícitamente para cubrir todas las eventualidades de modo que no sea necesaria la reapertura implícita. El cierre se ha impedido porque la cola se ha abierto con MQOO_SAVE_ALL_CONTEXT y anteriormente ya se había ejecutado una operación de obtención destructiva. Esto ha hecho que la cola tenga asociada información de estado retenido y el cierre podría destruir esta información.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Un objeto abierto no tiene las **OpenOptions** correctas y requiere una o más opciones adicionales. Se requiere una reapertura implícita, pero se ha impedido el cierre. Establezca **OpenOptions**explícitamente para cubrir todas las eventualidades de modo que no sea necesaria la reapertura implícita. Se ha impedido el cierre porque la cola es una cola local del tipo de definición MQQDT_TEMPORARY_DYNAMIC, que se destruiría por el cierre.
MQRC_ATTRIBUTE_LOCKED (6104)	Se ha intentado modificar el valor o atributo de un objeto mientras dicho objeto estaba abierto. Algunos atributos como, por ejemplo, **AlternateUserId**, no se pueden cambiar mientras un objeto está abierto.
MQRC_CURSOR_NOT_VALID (6105)	El cursor para examinar de una cola abierta ha quedado invalidado desde que una reapertura implícita lo utilizó por última vez. Establezca las OpenOptions explícitamente para cubrir todas las eventualidades de modo que no sea necesaria la reapertura implícita.
MQRC_ENCODING_ERROR (6106)	La codificación del siguiente elemento de mensaje debe ser MQENC_NATIVE para lectura.
MQRC_STRUCID_ERROR (6107)	La estructura del ID del siguiente elemento de mensaje, que se deriva de los 4 caracteres al comienzo del puntero de datos, falta o es incoherente con el tipo de la variable en la que se está leyendo el elemento.

Tabla 158. Códigos de razón y lo que significan (continuación)

Código de razón	Explicación
MQRC_NULL_POINTER (6108)	Se ha facilitado un puntero nulo y se requiere (o está implícito) un puntero que no sea nulo. Esto puede deberse al uso de declaraciones explícitas para objetos de WebSphere MQ utilizados desde VBA como parámetros para llamadas (por ejemplo, dim msg as Object is ok, dim msg as MqMessage puede causar problemas). Por ejemplo, en Excel, con q definido y establecido dim msg como MqMessageq.put msg proporciona reasonCode MQRC_NULL_POINTER. Funciona correctamente desde VisualBasic.
MQRC_NO_CONNECTION_REFERENCE (6109)	El objeto **MQQueue** ha perdido su conexión con el **MQQueueManager**. Esto se producirá si el **MQQueueManager**está desconectado. Borre el objeto **MQQueue**.
MQRC_NO_BUFFER (6110)	No hay ningún búfer disponible. Para un objeto **MQMessage** , no se puede asignar uno, lo que indica una incoherencia interna en el estado del objeto que no debería producirse.
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	La longitud de los datos binarios no es coherente con la longitud del atributo de destino. Cero es una longitud correcta para todos los atributos. 24 es una longitud correcta para un **CorrelationId** y para un **MessageId** 32 es una longitud correcta para un **AccountingToken**
MQRC_BUFFER_NOT_AUTOMATIC (6112)	Un búfer definido y gestionado por un usuario no puede redimensionarse. Como los búfers de mensajes están gestionados por el sistema, esto indica una incoherencia interna.
MQRC_INSUFFICIENT_BUFFER (6113)	No hay suficiente espacio disponible en el búfer después del puntero de datos para que quepa la petición. Esto puede deberse a que el búfer no puede redimensionarse.
MQRC_INSUFFICIENT_DATA (6114)	Después del puntero de datos, no queda suficiente espacio disponible en el búfer para dar cabida a la petición de lectura. Reduzca el búfer para que tenga el tamaño correcto y lea de nuevo los datos.
MQRC_DATA_TRUNCATED (6115)	Los datos se han truncado al copiarlos de un búfer a otro. Esto puede deberse a que el búfer de destino no puede redimensionarse o que hay un problema al direccionar uno de los dos búfers o a que se está reduciendo el tamaño de un búfer para una sustitución de menor tamaño.
MQRC_ZERO_LENGTH (6116)	Se ha facilitado una longitud cero y se requiere (o está implícita) una longitud positiva.
MQRC_NEGATIVE_LENGTH (6117)	Se ha facilitado una longitud negativa y se requiere una longitud cero o positiva.
MQRC_NEGATIVE_OFFSET (6118)	Se ha facilitado un desplazamiento negativo y se requiere un desplazamiento cero o positivo.
MQRC_INCONSISTENT_FORMAT (6119)	El formato del siguiente elemento de mensaje es incoherente con el tipo de la variable en la que está leyéndose el elemento.
MQRC_INCONSISTENT_OBJECT_STATE (6120)	Existe una incoherencia entre este objeto, que está abierto, y el objeto MQQueueManager referenciado, que no está conectado.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	La referencia de contexto de opciones MQPutMessageno hace referencia a un objeto MQQueue válido. El objeto ha sido destruido previamente.
MQRC_CONTEXT_OPEN_ERROR (6122)	La referencia de contexto de opciones MQPutMessagehace referencia a un objeto MQQueue que no se ha podido abrir para establecer un contexto. Esto puede deberse a que el objeto MQQueue tiene opciones de apertura inadecuadas. Examine el código de razón del objeto al que se hace referencia para determinar la causa.
MQRC_STRUC_LENGTH_ERROR (6123)	La longitud de una estructura de datos interna no es coherente con su contenido. Para una MQRMH, la longitud es insuficiente para contener los campos fijos y todos los datos de desplazamiento.

Tabla 158. Códigos de razón y lo que significan (continuación)	
Código de razón	**Explicación**
MQRC_NOT_CONNECTED (6124)	Un método ha fallado porque una conexión necesaria con un gestor de colas no estaba disponible y no se puede establecer una conexión implícitamente.
MQRC_NOT_OPEN (6125)	Un método ha fallado porque un objeto WebSphere MQ no estaba abierto y la apertura no se puede realizar implícitamente.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	No se ha podido abrir una MQDistributionList porque no hay ningún objeto de elemento MQDistributionList en la lista de distribución. Acción correctiva: añada al menos un objeto de elemento MQDistributionLista la lista de distribución.
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	Un método no se ha ejecutado correctamente porque el objeto está abierto y las opciones abiertas no son coherentes con la operación requerida. Acción correctiva: Abra el objeto con las opciones de apertura adecuadas y vuelva a intentarlo.
MQRC_WRONG_VERSION (6128)	Ha fallado un método porque un número de versión especificado o encontrado es incorrecto o no está soportado.

Herramienta de nivel de código

Es posible que el equipo de servicio de IBM le pregunte qué nivel de código ha instalado.

Para averiguarlo, ejecute la utilidad 'MQAXLEV'.

En un indicador de comandos, vaya al directorio que contenga MQAX200.dll o añada la ruta completa y ejecute:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

donde MQAXLEV.OUT es el nombre del archivo de salida.

Si no especifica un archivo de salida, el detalle se visualizará en la pantalla.

Un ejemplo de archivo de salida de la herramienta de nivel de código se detalla en el ejemplo siguiente:

Ejemplo de archivo de salida de la herramienta de nivel de código

```
5639-B43 (C) Copyright IBM Corp. 1996, 2024. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 ***** lib/mqole/mqole.cpp, mqole, p000, p000 L981119 1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216             1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212               1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212             1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212           1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212           1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212           1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219             1.11 99/02/18 12:12:59
```

Interfaz ActiveX con la MQAI

Para obtener una breve descripción general de las interfaces COM y su uso en la MQAI, consulte “Utilización de la interfaz de modelo de objeto de componente (WebSphere MQ Automation Classes for ActiveX)” en la página 1052.

La MQAI permite a las aplicaciones crear y enviar comandos en formato de comando programable (PCF) sin obtener y formatear directamente los búfers de longitud variable necesarios para PCF. Para obtener más información sobre la MQAI, consulte Introducción a WebSphere MQ Administration Interface (MQAI).

La clase MQAI ActiveX MQBag encapsula los paquetes de datos soportados por la MQAI de una forma que es posible utilizar en cualquier lenguaje que soporte la creación de objetos COM; por ejemplo, Visual Basic, C++, Java y otros clientes de scripts ActiveX .

La interfaz ActiveX de MQAI se utiliza con las clases MQAX que proporcionan una interfaz COM a la MQI. Para obtener más información sobre las clases MQAX, consulte “Diseño de aplicaciones MQAX que acceden a aplicaciones que no son de ActiveX” en la página 1053.

La interfaz de ActiveX proporciona una única clase llamada MQBag. Esta clase se utiliza para crear paquetes de datos MQAI, y sus propiedades y métodos se utilizan para crear elementos de datos dentro de cada paquete y trabajar con ellos. El método MQBag Execute envía los datos de paquete a un gestor de colas de WebSphere MQ como un mensaje PCF y recopila las respuestas.

Para obtener más información sobre la clase MQBag, sus propiedades y métodos, consulte “La clase MQBag” en la página 1130.

El mensaje PCF se envía al objeto del gestor de colas especificado, opcionalmente utilizando las colas de solicitud y respuesta especificadas. Las respuestas se devuelven en un nuevo objeto MQBag. El conjunto completo de comandos y respuestas se describe en Definiciones de los formatos de comandos programables. Los mandatos se pueden enviar a cualquier gestor de colas de la red de WebSphere MQ seleccionando las colas de solicitudes y respuestas adecuadas.

La clase MQBag

La clase MQBag se utiliza para crear los objetos MQBag necesarios. Cuando se ha iniciado, la clase MQBag devuelve una nueva referencia al objeto MQBag.

Cree un objeto MQBag en Visual Basic según se indica a continuación:

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

Propiedad MQBag

Las propiedades de los objetos MQBag se explican en la lista siguiente:

- “Propiedad Item” en la página 1131.
- “propiedad Count” en la página 1132.
- “Propiedad Options” en la página 1132.

Métodos MQBag

Los métodos de los objetos MQBag se explican en la lista siguiente:

- “método Add” en la página 1133.
- “método AddInquiry” en la página 1134.
- “Método Clear” en la página 1134.
- “método Execute” en la página 1134.
- “método FromMessage” en la página 1135.
- “Método ItemType” en la página 1135.
- “método Remove” en la página 1136.
- “Método Selector” en la página 1137.
- “método ToMessage” en la página 1137.
- “método Truncate” en la página 1138.

Manejo de errores

Si se detecta un error durante una operación en un objeto MQBag, incluidos los errores devueltos al paquete por un objeto MQAX o MQAI subyacente, se producirá un error de excepción. La clase MQBag tiene soporte para la interfaz ISupportErrorInfo COMO, por lo que la siguiente información estará disponible para la rutina de manejo de errores.

- Número de error: compuesto por el código de razón WebSphere MQ para el error detectado y un código de recurso COM. El campo de recurso, que es un estándar de COM, indica el área de responsabilidad del error. Para los errores detectados por WebSphere MQ siempre es FACILITY_ITF.
- Origen de error: identifica el tipo y la versión del objeto que ha detectado el error. Para los errores detectados durante las operaciones MQBag, el origen de error es siempre MQBag.MQBag1.
- Descripción del error: la serie que proporciona el nombre simbólico para el código de razón WebSphere MQ .

El acceso a la información de error depende del lenguaje de scripts; por ejemplo, en Visual Basic la información se devuelve en el objeto Err y el código de razón WebSphere MQ se obtiene restando la constante vbObjectError de Err.Number.

ReasonCode = Err.Number - vbObjectError

Si el mensaje de ejecución de MQBag envía un mensaje PCF y se recibe una respuesta, la operación se considera satisfactoria aunque el mandato enviado podría haber fallado. En este caso, el propio paquete de respuesta contiene los códigos de razón de finalización y error tal como se describe en Definiciones de los formatos de mandato programable .

Propiedad Item

Finalidad

La propiedad Item representa un elemento de un paquete. Se utiliza para definir o consultar el valor de un elemento. El uso de esta propiedad corresponde a las llamadas MQAI siguientes:

- "mqSetString"
- "mqSetInteger"
- "mqInquireInteger"
- "mqInquireString"
- "mqInquireBag"

en Guía de consulta sobre formatos de mandatos programables.

Formato

Item (Selector, ItemIndex, Value)

Parámetros

Selector **(VARIANT)-entrada**
Selector del elemento que va a definirse o consultarse.

Cuando se consulta un elemento, el valor predeterminado es MQSEL_ANY_USER_SELECTOR. Cuando se establece un elemento, el valor predeterminado es MQIA_LIST o MQCA_LIST.

Si `Selector` no es de tipo Long, se genera MQRC_SELECTOR_TYPE_ERROR.

Este parámetro es opcional.

ItemIndex **(LONG)-entrada**
Este valor identifica la aparición del elemento del selector especificado que se ha de definir o consultar. MQIND_NONE es el valor predeterminado.

Este parámetro es opcional.

***Value* (VARIANT)-entrada/salida**
El valor devuelto o el valor que se va a definir. Cuando se efectúa una consulta sobre un elemento, el valor devuelto puede ser de tipo Long, String o MQBag. No obstante, cuando se establece un elemento, el valor debe ser de tipo Long o String. De lo contrario se genera MQRC_ITEM_VALUE_ERROR.

Invocación de lenguaje Visual Basic

Cuando se consulta un valor de un elemento incluido en un paquete:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

Para referencias de MQBag:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

Para establecer el valor de un elemento en un paquete:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

propiedad Count

Finalidad

La propiedad Count representa el número de elementos de datos que hay dentro de un paquete. Esta propiedad corresponde a la llamada MQAI "mqCountItems" en Guía de consulta sobre formatos de mandatos programables.

Formato

Recuento (*Selector, Value*)

Parámetros

***Selector* (VARIANT)-entrada**
Selector de los elementos de datos que deben incluirse en el recuento.

MQSEL_ALL_USER_SELECTORS es el valor predeterminado.

Si el parámetro `Selector` no es de tipo long, se devuelve MQRC_SELECTOR_TYPE_ERROR.

***Value* (LONG)-salida**
El número de elementos en el paquete incluido por *Selector*.

Invocación de lenguaje Visual Basic

Para devolver el número de elementos de un paquete:

```
ItemCount = mqbag.Count([Selector])
```

Propiedad Options

Finalidad

La propiedad Options define las opciones para utilizar un paquete. Esta propiedad se corresponde al parámetro `Options` de la llamada MQAI "mqCreateBag" en Guía de consulta sobre formatos de mandatos programables.

Formato

Opciones (*`Options`*)

Parámetros

***`Options`* (LONG)-entrada/salida**
Las opciones del paquete.

Nota: Las opciones del paquete deben establecerse ***antes*** de que se añadan o se establezcan elementos de datos dentro del paquete. Si las opciones se modifican cuando el paquete no está vacío, se genera MQRC_OPTIONS_ERROR. Esto es así aunque el paquete se borre a continuación.

Invocación de lenguaje Visual Basic

Cuando se consultan las opciones de un elemento dentro de un paquete:

```
Options = mqbag.Options
```

Para establecer una opción de un elemento en un paquete:

```
mqbag.Options = Options
```

Métodos MQBag

Los métodos de los objetos MQBag se explican en las páginas siguientes.

método Add

Finalidad

El método Add añade un elemento de datos a un paquete. Este método corresponde a las llamadas MQAI "mqAddInteger" y "mqAddString" en Guía de consulta sobre formatos de mandatos programables.

Formato

Añadir (*`Value, Selector`*)

Parámetros

***`Value`* (VARIANT)-entrada**
Valor entero o de serie del elemento de datos.

***`Selector`* (VARIANT)-entrada**
Selector que identifica el elemento que se va a añadir.

Dependiendo del tipo de `Value`, MQIA_LIST o MQCA_LIST es el valor predeterminado. Si el parámetro `Selector` no es de tipo Long, se genera MQRC_SELECTOR_TYPE_ERROR.

Invocación de lenguaje Visual Basic

Para añadir un elemento a un paquete:

```
mqbag.Add(Value,[Selector])
```

método AddInquiry

Finalidad

El método AddInquiry añade un selector especificando el atributo que debe devolverse cuando se envía un paquete de administración para ejecutar un mandato INQUIRE. Este método corresponde a la llamada MQAI "mqAddInquiry" en Guía de consulta sobre formatos de mandatos programables.

Formato

AddInquiry *(Inquiry)*

Parámetros

***Inquiry* (LONG)-entrada**
Selector del atributo WebSphere MQ que debe devolver el mandato de administración INQUIRE.

Invocación de lenguaje Visual Basic

Para utilizar el método AddInquiry:

```
mqbag.AddInquiry(Inquiry)
```

Método Clear

Finalidad

El método Clear suprime todos los elementos de datos de un paquete. Este método corresponde a la llamada MQAI "mqClearBag" en Guía de consulta sobre formatos de mandatos programables.

Formato

Borrar

Invocación de lenguaje Visual Basic

Para suprimir todos los elementos de datos de un paquete:

```
mqbag.Clear
```

método Execute

Finalidad

El método Execute envía un mensaje de mandato de administración al servidor de mandatos y espera los mensajes de respuesta. Este método corresponde a la llamada MQAI "mqExecute" en Guía de consulta sobre formatos de mandatos programables.

Formato

Ejecute *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*)

Parámetros

QueueManager **(MQQueueManager)-entrada**
El gestor de colas al que se conecta la aplicación.

Command **(LONG)-entrada**
El mandato que se va a ejecutar.

OptionsBag **(MQBag)-entrada**
El paquete que contiene las opciones que afectan al proceso de la llamada.

RequestQ **(MQQueue)-entrada**
La cola donde se colocará el mensaje del mandato de administración.

ReplyQ **(MQQueue)-entrada**
La cola en la que se reciben los mensajes de respuesta.

ReplyBag **(MQBag)-salida**
Referencia a un paquete que contiene datos de los mensajes de respuesta.

Invocación de lenguaje Visual Basic

Para enviar un mensaje de mandato de administración y esperar los mensajes de respuesta:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

método FromMessage

Finalidad

El método FromMessage carga en un paquete datos de un mensaje. Este método corresponde a la llamada MQAI "mqBufferToBag" en Guía de consulta sobre formatos de mandatos programables.

Formato

FromMessage *(Message, OptionsBag*)

Parámetros

Message **(MQMessage)-entrada**
El mensaje que contiene los datos que van a convertirse.

OptionsBag **(MQBag)-entrada**
Opciones para controlar el proceso de la llamada.

Invocación de lenguaje Visual Basic

Para cargar datos de un mensaje en un paquete:

```
mqbag.FromMessage(Message,[OptionsBag])
```

Método ItemType

Finalidad

El método ItemType devuelve el tipo de valor en un elemento especificado de un paquete. Este método corresponde a la llamada MQAI, "mqInquireItemInfo,", en Guía de consulta sobre formatos de mandatos programables.

Formato

ItemType *(Selector, ItemIndex, ItemType*)

Parámetros

***Selector* (VARIANT)-entrada**
Selector que identifica el elemento que va a consultarse.

MQSEL_ANY_USER_SELECTOR es el valor predeterminado. Si el parámetro `Selector` no es de tipo Long, se genera MQRC_SELECTOR_TYPE_ERROR.

***ItemIndex* (LONG)-entrada**
Índice de los elementos a consultar.

MQIND_NONE es el valor predeterminado.

***ItemType* (LONG)-salida**
El tipo de datos del elemento especificado.

Nota: Deben especificarse el parámetro `Selector`, el parámetro `ItemIndex` o ambos. Si no está presente ninguno de estos parámetros, se genera MQRC_PARAMETER_MISSING.

Invocación de lenguaje Visual Basic

Para devolver el tipo de un elemento:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

método Remove

Finalidad

El método Remove suprime un elemento de un paquete. Este método corresponde a la llamada MQAI "mqDeleteItem" en Guía de consulta sobre formatos de mandatos programables.

Formato

Elimine *(Selector, ItemIndex*)

Parámetros

***Selector* (VARIANT)-entrada**
Selector que identifica el elemento que va a suprimirse.

MQSEL_ANY_USER_SELECTOR es el valor predeterminado. Si el parámetro `Selector` no es de tipo Long, se genera MQRC_SELECTOR_TYPE_ERROR.

***ItemIndex* (LONG)-entrada**
Índice del elemento que va a suprimirse.

MQIND_NONE es el valor predeterminado.

Nota: Deben especificarse el parámetro `Selector`, el parámetro `ItemIndex` o ambos. Si no está presente ninguno de estos parámetros, se genera MQRC_PARAMETER_MISSING.

Invocación de lenguaje Visual Basic

Para suprimir un elemento de un paquete:

```
mqbag.Remove([Selector],[ItemIndex])
```

Método Selector

Finalidad

El método Selector devuelve el selector de un elemento especificado dentro de un paquete. Este método corresponde a la llamada MQAI, "mqInquireItemInfo,", en Guía de consulta sobre formatos de mandatos programables.

Formato

Selector ***(Selector, ItemIndex, OutSelector)***

Parámetros

Selector **(VARIANT)-entrada**
Selector que identifica el elemento que va a consultarse.

MQSEL_ANY_USER_SELECTOR es el valor predeterminado. Si el parámetro `Selector` no es de tipo Long, se genera MQRC_SELECTOR_TYPE_ERROR.

ItemIndex **(LONG)-entrada**
Índice del elemento que debe consultarse.

MQIND_NONE es el valor predeterminado.

OutSelector **(VARIANT)-salida**
Selector del elemento especificado.

Nota: Deben especificarse el parámetro `Selector`, el parámetro `ItemIndex` o ambos. Si no está presente ninguno de estos parámetros, se genera MQRC_PARAMETER_MISSING.

Invocación de lenguaje Visual Basic

Para devolver el selector de un elemento:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

método ToMessage

Finalidad

El método ToMessage devuelve una referencia a un objeto MQMessage. La referencia contiene datos de un paquete. Este método corresponde a la llamada MQAI "mqBagToBuffer" en Guía de consulta sobre formatos de mandatos programables.

Formato

ToMessage *(OptionsBag, Message)*

Parámetros

OptionsBag **(MQBag)-entrada**
Un paquete que contiene las opciones que controlan el proceso del método.

***Message* (MQMessage)-salida**
Una referencia a un objeto MQMessage que contiene datos del paquete.

Invocación de lenguaje Visual Basic

Para utilizar el método ToMessage:

```
Set Message = mqbag.ToMessage([OptionsBag])
```

método Truncate

Finalidad

El método Truncate reduce el número de elementos de usuario en un paquete. Este método corresponde a la llamada MQAI "mqTruncateBag" en Guía de consulta sobre formatos de mandatos programables.

Formato

Truncar (*ItemCount*)

Parámetros

***ItemCount* (LONG)-entrada**
El número de elementos de usuario que debe permanecer en el paquete después de que se haya producido el truncamiento.

Invocación de lenguaje Visual Basic

Para reducir el número de elementos de usuario en una paquete:

```
mqbag.Truncate(ItemCount)
```

Acerca de los ejemplos de WebSphere MQ Automation Classes for ActiveX Starter

En este apéndice se describen los ejemplos de WebSphere MQ Automation Classes for ActiveX Starter y se explica cómo utilizarlos.

WebSphere MQ for Windows proporciona los siguientes programas de ejemplo de Visual Basic:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

Estos ejemplos se ejecutan en Visual Basic 4 o Visual Basic 5. Los encontrará en el directorio ... \tools\mqax\samples\vb.

En el mismo directorio también encontrará ejemplos para Microsoft Excel y html. Son las siguientes:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

Nota: Si utiliza Visual Basic 5, **deberá** seleccionar e instalar el componente de Visual Basic grid32.ocx.

¿Qué se muestra en los ejemplos?

Los ejemplos muestran cómo utilizar WebSphere MQ Automation Classes for ActiveX para:

- Conectarse a un gestor de colas
- Acceder a una cola
- Colocar un mensaje en una cola
- Obtener un mensaje de una cola

La parte central del ejemplo de Visual Basic se muestra en las siguientes páginas.

"Preparación de la ejecución de los ejemplos" en la página 1139and

"Tratamiento de errores en los ejemplos" en la página 1140

ejecución de los ejemplos iniciales de ActiveX

Antes de ejecutar los ejemplos de WebSphere MQ Automation Classes for ActiveX Starter, compruebe que tiene un gestor de colas predeterminado en ejecución y que ha creado las definiciones de cola necesarias. Encontrará información detallada sobre la creación y ejecución de un gestor de colas y la creación de una cola en la publicación Administración. El ejemplo utiliza la cola SYSTEM.DEFAULT.LOCAL.QUEUE que se debe definir en cualquier servidor WebSphere MQ configurado normalmente.

Las distintas formas de utilizar paquetes de datos son las que se muestran en la lista siguiente:

- Conectarse a un gestor de colas
- Acceder a una cola
- Colocar un mensaje en una cola
- Obtener un mensaje de una cola

Para obtener información sobre los ejemplos de inicio de MQAX para Microsoft Basic Versión 4 o posterior, consulte "Ejecución del ejemplo MQAXTRIV" en la página 1140

Para obtener información sobre un ejemplo que le permita examinar las propiedades y los métodos de gestores de colas y objetos de cola, consulte "Inicio del ejemplo MQAXCLSS" en la página 1141

Para obtener información sobre el ejemplo MQAXDLST, "El ejemplo MQAXDLST" en la página 1142

Para obtener información sobre cómo ejecutar el ejemplo de inicio MQAX para Microsoft Excel 95 o posterior, MQAXTRIV.XLS, consulte "Ejecución del ejemplo MQAXTRIV.XLS" en la página 1142.

Para obtener información sobre la ejecución de la demostración del banco (Bank) con MQAX.XLS, consulte "Ejecución de la demostración del banco con MQAX.XLS" en la página 1142

Para obtener información sobre el ejemplo de iniciador utilizando un navegador de WWW compatible con ActiveX, consulte "Ejemplo inicial utilizando un navegador WWW compatible con ActiveX" en la página 1142

Preparación de la ejecución de los ejemplos

Para ejecutar cualquiera de los ejemplos, se necesita una de las siguientes opciones en función del ejemplo que se quiera ejecutar.

- Microsoft Visual Basic Versión 4 (o posterior)
- Microsoft Excel 95 (o posterior)
- Un navegador web.

También se necesita:

- Un gestor de colas de WebSphere MQ en ejecución.
- Ya se ha definido una cola de WebSphere MQ .

Tratamiento de errores en los ejemplos

La mayoría de los ejemplos proporcionados en el paquete WebSphere MQ Automation Classes for ActiveX muestran poco o ningún manejo de errores. Para obtener más información sobre el tratamiento de errores, consulte “Tratamiento de errores” en la página 1057.

Ejecución del ejemplo MQAXTRIV

1. Inicie el gestor de colas.
2. En Windows Explorer o File Manager, seleccione el icono para el ejemplo, MQAXTRIV.VBP (archivo de proyecto de Visual Basic) y abra el archivo.

   Se iniciará el programa Visual Basic y abrirá el archivo MQAXTRIV.VBP.
3. En Visual Basic, pulse la tecla de función 5 (F5) para ejecutar el ejemplo.
4. Pulse en cualquier lugar del formulario de ventana, "MQAX trivial tester".

Si todo funciona correctamente, el fondo de la ventana cambiará a verde. Si hay algún problema en la instalación, el fondo de la ventana cambiará a rojo y se visualizará información de error.

La figura siguiente muestra la parte central del ejemplo en Visual Basic.

```
Option Explicit

Private Sub Form_Click()

'*****************************************************************************
'* This simple example illustrates how to put and get a WebSphere MQ message to
'* and from a WebSphere MQ message queue. The data from the message returned by the
'*get is read and compared with that from the original message.
'*****************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* get message option

Dim GetOptions As MQGetMessageOptions   '* put message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*****************************************************************************
'* Handle errors
'*****************************************************************************
On Error GoTo HandleError

'*****************************************************************************
'* Initialize the current position for the form
'*****************************************************************************
CurrentX = 0
CurrentY = 0

'*****************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*****************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                        MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*****************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*****************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*****************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
```

```
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*******************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*******************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*******************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127)  '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*******************************************************************************
'* Handle errors
'*******************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
    Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error:  " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

Inicio del ejemplo MQAXCLSS

Este ejemplo permite examinar propiedades y métodos de objetos de cola y gestores de colas.

1. Inicie el gestor de colas.
2. Abra el archivo, MQAXCLSS.VBP, haciendo doble clic en el icono de documento en Windows Explorer o haciendo clic en Archivo-Abrir en el menú de archivo en Visual Basic.
3. Inicie el ejemplo.
4. Especifique los nombres de cola y gestor de colas adecuados y pulse los botones correspondientes.

El ejemplo MQAXDLST

El ejemplo de Visual Basic MQAXDLST ilustra el uso de una lista de distribución para enviar el mismo mensaje a dos colas con una sola colocación. Para ejecutar el ejemplo, haga lo mismo que en el ejemplo MQAXCLSS.

Ejemplo de iniciador MQAX para Microsoft Excel 95 o posterior

En esta sección se explica cómo ejecutar el ejemplo de inicio MQAX para Microsoft Excel 95 o posterior, MQAXTRIV.XLS.

Ejecución del ejemplo MQAXTRIV.XLS

1. Inicie el gestor de colas.
2. En el Explorador o en el Administrador de archivos, seleccione el icono del ejemplo de MQAX, MQAXTRIV.XLS.
3. Pulse el botón en la hoja de cálculo.
4. La pantalla se actualiza con un mensaje de éxito (o error).

Ejecución de la demostración del banco con MQAX.XLS

Siga estos pasos para ejecutar la demostración del banco.

1. Inicie el gestor de colas.
2. Ejecute el archivo de mandatos MQSC de IBM WebSphere MQ, BANK.TST. Esto configura las definiciones de cola de IBM WebSphere MQ necesarias.

   Para obtener información sobre cómo utilizar un archivo de mandatos MQSC, consulte Mandatos de script (MQSC).
3. Ejecute MQAXBSRV.VBP. Este programa de ejemplo es el servidor que simula una aplicación de programa de fondo, y se tiene que ejecutar con Microsoft Excel.
4. Ejecute MQAX.XLS. Este ejemplo es la demostración del cliente IBM WebSphere MQ .
5. Seleccione un cliente de la lista.
6. Pulse **Enviar**.

Después de una pausa de unos 3 segundos, los campos se llenan con valores y se muestra un diagrama de barras.

Ejemplo inicial utilizando un navegador WWW compatible con ActiveX

Nota: Para ejecutar este ejemplo, debe ejecutar un navegador web compatible con ActiveX. Microsoft Internet Explorer (pero no Netscape Navigator) es un navegador web compatible.

Ejecución del ejemplo HTML

Este ejemplo muestra cómo puede invocar MQAX desde VBScript y JavaScript.

1. Inicie el gestor de colas.
2. Abra el archivo "MQAXTRIV.HTM" en su navegador web compatible con ActiveX.

   Puede hacerlo efectuando una doble pulsación en el icono de archivo en Windows Explorer o puede elegir Archivo-Abrir en el menú Archivo de su navegador web compatible con ActiveX .
3. Siga las instrucciones de la pantalla.

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 disponibles actualmente en su zona. Las referencias a programas, productos o servicios de IBM no pretenden indicar ni implicar que sólo puedan utilizarse los 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:

Licencias de Propiedad Intelectual
Ley de Propiedad intelectual y legal
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 efectuar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento y 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

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, algunas mediciones pueden haberse estimado por 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. Las preguntas relacionadas con las posibilidades de los productos que no sean 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 COPYRIGHT:

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 IBM 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 EE.UU. y/o en 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 (http://www.eclipse.org/).

Java y todas las marcas registradas y logotipos son marcas registradas de Oracle o sus afiliados.

Número Pieza:

(1P) P/N: